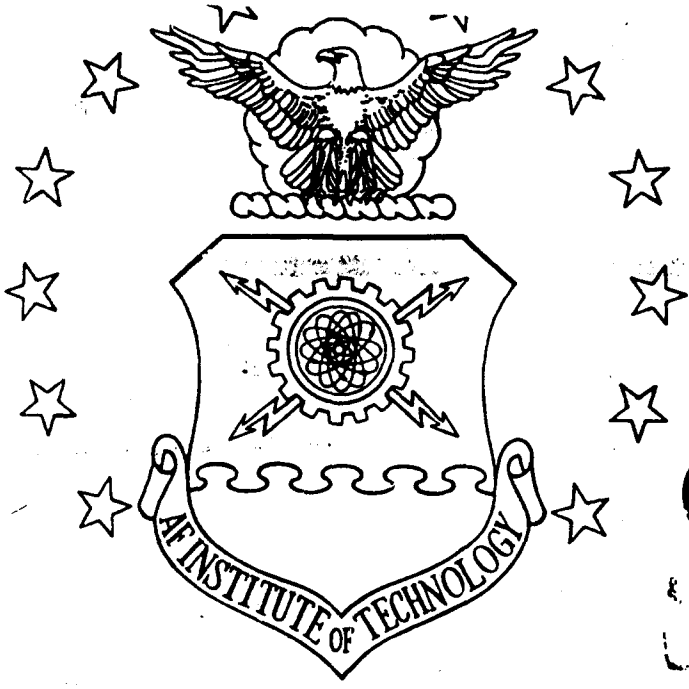


DEPARTMENT OF THE AIR FORCE

AIR UNIVERSITY

AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

AFIT GCS/ENG/91D-13

ESTABLISHING A METHODOLOGY FOR EVALUATING AND SELECTING COMPUTER AIDED SOFTWARE ENGINEERING TOOLS FOR A DEFINED SOFTWARE ENGINEERING ENVIRONMENT AT THE AIR FORCE INSTITUTE OF TECHNOLOGY SCHOOL OF ENGINEERING

THESIS

Jody L. Mattingly, Captain, USAF

AFIT GCS/ENG/91D-13

AFIT GCS/ENG/91D-13

Establishing a Methodology for Evaluating and Selecting Computer Aided Software Engineering Tools for a Defined Software Engineering Environment at the Air Force Institute of Technology School of Engineering

THESIS

Presented to the Faculty of the School of Engineering
of the Air Force Institute of Technology
Air University
In Partial Fulfillment of the
Requirements for the Degree of
Master of Computer Science in Computer Systems

Jody L. Mattingly
Captain, USAF

December, 1991

Approved for public release; distribution unlimited

Preface

The purpose of this thesis is to identify the software engineering environment (SEE) as it exists at the Air Force Institute of Technology School of Engineering, determine the software process model employed and the software development methods presented as part of the curriculum, and then, based on this information, develop criteria upon which to evaluate computer aided software engineering (CASE) tools being considered for integration into the SEE, evaluate these CASE tools using this criteria, analyze the results and make recommendations concerning the SEE and CASE tools based on this analysis.

I extend my gratitude to several people who supported me during this effort. I thank my thesis advisor, Lt Col Patricia Lawlis for her guidance and expertise in the area of SEEs. I also acknowledge the support provided me by my thesis committee, Maj Paul Bailor and Maj Jim Howatt. I greatly appreciated their insights and suggestions. I would also like to thank Maj Roger Koble, Capt Mike Dedolph, Capt Dawn Guido, and Capt Karen Harrower for their assistance with the CASE tool evaluations. I thank Capt Dean Campbell, Mr. Dan Zambon and Mr. Doug Burkholder for their assistance concerning the computer systems which hosted the CASE tools under consideration. Most especially, I would like to express my gratitude to my wife, Karen, for the patience, support, encouragement, and understanding she provided during this effort.

Table of Contents

List of Figures

List of Tables

Acronyms

AFIT - Air Force Institute of Technology.

APSE - Ada Programming Support Environment.

CASE - Computer Aided Software Engineering.

CDD - Common Data Dictionary.

CMS - Code Management System.

DEC - Digital Equipment Corporation.

DTM - DEC Test Manager.

GKS - Graphical Kernel System.

IBM - International Business Machines.

ICASE - Integrated Computer Aided Software Engineering.

IPSE - Integrated Program Support Environment.

KAPSE - Kernel Ada Programming Support Environment.

LSE - Language Sensitive Editor.

MAPSE - Minimal Ada Programming Support Environment.

MMS - Module Management System.

OS - Operating System.

PC - Personal Computer.

PCA - Performance and Coverage Analyzer.

PCE - Professional Continuing Education.

PHIGS - Programmer's Hierarchical Interactive Graphics System.

POSE - Picture Oriented Software Engineering.

RDF - Rational Design Facility.

SEE - Software Engineering Environment.

SQL - Structured Query Language.

VAX - Virtual Address Extension.

VMS - Virtual Memory System.

Abstract

This thesis identifies the software engineering environment (SEE) as it exists at the Air Force Institute of Technology (AFIT) School of Engineering. It also describes the software process model employed and the software development methods presented as part of the curriculum. Based on this information, criteria was established to evaluate computer aided software engineering (CASE) tools being considered for integration into the SEE. Each criterion was weighted to indicate its importance when selecting CASE tools. The criteria were further used to establish a methodology to be used to evaluate and select the CASE tools under consideration as well as future tool candidates.

I. Introduction

Recently, the Air Force Institute of Technology (AFIT) had a number of Computer Aided Software Engineering (CASE) tool sets available which required evaluation to determine their potential use, if any, at AFIT. Three particular CASE tools were being considered. Neither staff nor students were typically familiar with any of these tools, although the various analysis and design methodologies the tools incorporate are taught as part of the curriculum and should be familiar to both staff and students.

A coordinated and useable Software Engineering Environment (SEE) that is appropriate for AFIT needs in the area of software development had to first be defined. An established SEE would enhance the productivity of those who work with the environment. The defined environment was based on existing and applicable AFIT hardware, as well as software which executes on these systems. The SEE should be expandable, and provide good software engineering support.

Once the environment was defined, CASE tools with the potential to be an integral part of the SEE were identified. The criteria upon which to evaluate these CASE tools was then established, followed by an evaluation and analysis of the tools. After this had been accomplished, the SEE (or a prototype) was built.

Problem Definition

A SEE which supports software development activities at AFIT needed to be defined so that it could be determined how CASE tools under consideration for use at AFIT might be incorporated into this environment. An investigation of existing methods and measurements developed for the purpose of evaluating and selecting CASE tools was required, and from this collection of information, a determination was made of the appropriate criteria upon which to evaluate CASE tools for this environment.

Research Objectives

The primary objectives of this thesis were as follows:

1. To define a SEE in which the proposed CASE tools would be incorporated.

2. To establish criteria with which to evaluate the three CASE tools under consideration based on the SEE and the software development methodologies presented at AFIT.

3. To determine how each CASE tool supported the chosen criteria.

4. To investigate existing methods and measurements used to evaluate CASE tools.

5. To determine how the CASE tools compared to each other using the chosen evaluation method.

6. To determine the usefulness of each to AFIT staff and students, and if any tool should be incorporated in the SEE.

Research Questions

Five key questions were addressed by this thesis:

1. 'How can a software engineering environment at AFIT be defined?' This question pertained to the first two objectives and needed to be answered before all others. A thorough understanding of the SEE aided in the determination of the role of CASE tools in that environment.

2. 'What criteria should be used to evaluate the CASE tools under consideration?' This question related to the second objective of the thesis. There are a great number of criteria against which CASE tools can be evaluated and selected. The criteria chosen were the most important which apply to the problem at hand, while less important criteria were disregarded.

3. 'How do the CASE tools under consideration meet the established criteria for each?' This question corresponds to Objective 3. The criteria for each tool was not necessarily the same criteria as established for the other tools, although they were similar. Each tool was judged first by the criteria established for that particular tool.

4. 'How do the tools under consideration compare, if at all, to each other?' This question relates to the fourth and fifth objectives. The tools were examined to determine if the capabilities provided by each are similar to the other. Certainly, if the tools provided the same or very similar capabilities, a determination would be made as to which tool should be chosen over all others. On the other hand, it may

be determined that each tool provides enough unique capabilities that all tools will be chosen for use at AFIT. In order to make these types of determinations, a methodology for evaluating and selecting CASE tools was first decided upon.

5. 'Do these tools belong in the software engineering environment at AFIT, and if so, how would these tools be incorporated into the defined environment?' This question corresponds to Objective 6. The possibility existed that after evaluation of the CASE tools, it would be determined that one or more would not be useful at AFIT. Any tool that was determined to be useful would have to be examined further to determine how it would be integrated into the SEE.

Approach

The first step in the thesis research was to determine how to define the SEE in which the proposed CASE tools would exist. This required a literature search on the topic of Software Environments and a thorough study of the accumulated material. Once this research was complete, a SEE for AFIT, of which the CASE tools were being considered as part, was defined.

Next, a survey of literature related to the topic of CASE tool evaluation was required. Based on this research, as well as consultations with AFIT staff who are directly concerned with the evaluation of the CASE tools, establishment of the criteria upon which to evaluate these tools was accomplished.

Then, an initial examination of all the tools was performed to provide familiarity with both the tools and the environments they execute in. The tools were then evaluated, beginning with the least complicated as a way of becoming familiar with the evaluation process. As the tools were evaluated, personnel experienced in CASE tool evaluation were conferred with for their expertise. For each tool, a test problem was established to assist in the evaluation process. The CASE tools that were evaluated were three sets which are available at AFIT, Picture Oriented Software Engineering (POSE), DECdesign, and ObjectMaker.

POSE, developed by Computer System Advisors, provides the user the ability to use structured analysis and design techniques to develop a software project. A limited evaluation on this CASE tool had been accomplished prior to this thesis for the purpose of determining its potential applications for Object Oriented Analysis and Design. As part of the thesis, a more comprehensive evaluation of POSE was performed, based on the criteria established as a result of the research on CASE evaluation methods. POSE is currently resident on the AFIT personal computer (PC) network.

The second tool set evaluated was DECdesign, created and distributed by Digital Equipment Corporation (DEC). DECdesign executes in a VAX environment and also provides the user the ability to use structured analysis and design techniques. The tool is currently loaded on a local VAX.

The third tool set evaluated was ObjectMaker, developed by Mark V Systems. This tool set provides the capabilities for structured analysis and design, object oriented analysis and design, generation of Ada and C code based on previously created designs, and reverse engineering. The tool set will execute in a PC, work station, or main frame environment. The tool is currently loaded on the AFIT PC network and is accessible using Zenith 248 machines.

Based upon established criteria for measuring and evaluating these tools, as well as input from AFIT staff as to how these tools would best fit into the AFIT academic environment, a determination was made if any or all of the CASE tools would be useful by both AFIT staff and students. This determination was also based upon the experiences and opinions of students (including this researcher) and staff who have tested the tool. Final analysis includes recommendations for further studies.

Materials and Equipment Required

Literature on defining software environments, as well as the evaluation of CASE tools, was required. This literature was obtained from the AFIT library. Other local university libraries were referenced, including Wright State University and the University of Dayton.

Documentation on the CASE tools under consideration was required. This included user's manuals, installation guides, tutorials, etc.

A VAX was required for the implementation of the DECdesign CASE tool, as well as access to the VAX (an account), and a terminal that would support DECdesign's graphic capabilities. Currently, DECdesign is installed on the local VAX and is accessible via the Digital work station in the AFIT graphics lab.

For the POSE tool, an IBM PC compatible environment, and access to the PC were required. When POSE was installed on the AFIT network, memory problems existed with the 286 based PCs. These problems prohibited the execution of some of the modules provided by POSE. These memory problems were rectified with the introduction of 386 based PCs into the computer inventory at AFIT.

ObjectMaker required a PC environment with Microsoft Windows version 3.0 and a mouse (or its equivalent). The demonstration copy was loaded on a Zenith 248. Because of the limited memory available on this machine, tool execution time was extremely slow.

Printers and/or plotters had to also be available to allow for hard copy output of graphical objects and documents created by the CASE tools.

Documentation on the systems used at AFIT was required as part of the evaluation of the SEE.

Finally, the CASE tools themselves needed to be available and resident in the AFIT software environment. Sample diskettes would not suffice for evaluation purposes. The tools needed to be available for a thorough and comprehensive

evaluation. POSE and DECdesign were available for the duration of this research. However, ObjectMaker was very new and available for only a short time at the very end of the research effort. The researcher was only able to test some design capabilities of ObjectMaker and conduct a limited evaluation.

Scope

The purpose of the research on SEEs was to define the environment in which the CASE tools under consideration might reside. The SEE research was therefore as detailed as time permitted and complete enough so that the roles of the CASE tools in relationship to this environment could be defined. The evaluation and building of a SEE at AFIT had the potential to be a thesis topic in itself. The main thrust of this thesis was the evaluation of the three CASE tools.

The criteria upon which the CASE tools were be evaluated, as well as the test case that was used, also had to be limited. The test case was designed so that it tested the features of the tools that would be of most interest to the AFIT community. Therefore, an exhaustive evaluation of the tools was not feasible or necessary.

Introduction to Remainder of the Thesis

Chapter 2 documents the results of the literature reviews and provides background material related to this thesis. The emphasis is on CASE tool evaluation methods and criteria for measuring tools.

Chapter 3 outlines the solution to the problem being examined. A detailed description of the AFIT SEE, as well as the criteria upon which the CASE tools were evaluated, the steps required to implement the evaluations, a description of the test cases used, and an indication of the resources/tools needed to solve the problem are provided.

Chapter 4 describes the actual implementation of the evaluations. Descriptions of the CASE tool evaluations, the evaluators, and the test cases used are provided.

Chapter 5 includes the analysis of the evaluations. Results of the evaluations (i.e., results of implementing a methodology for evaluating and selecting CASE tools) and their meaning are discussed.

Chapter 6 includes any conclusions about the thesis effort as well as recommendations for further study.

The appendices of the thesis contain a general description of the three CASE tools being evaluated, examples of the evaluations used, a listing of the criteria established to evaluate the CASE tools, any reports and any charts or graphics generated by the CASE tools.

II Literature Review

This chapter begins with an examination of SEEs. Various descriptions of a SEE are examined to determine a SEE's primary attributes. Benefits of establishing a SEE are then presented. Finally, various SEE types are presented, as well as software environment architectures.

The second part of this chapter deals with establishing criteria for evaluating CASE tools to be used in a SEE. First, a definition of CASE is presented. Next, CASE tool environments, types, and characteristics are examined. Finally, the benefits and potential pitfalls of utilizing CASE tools are considered, followed by an examination of criteria used to evaluate CASE tools.

2.1 Description of a SEE

Various authors have provided descriptions of what a SEE is and what a SEE should provide. Some of these descriptions are:

> A SEE assists the accomplishment of software engineering through sets of computer facilities, integrated software tools, and uniform engineering procedures. (61:384)
>
> A software engineering environment surrounds the user with tools needed to systematically develop and maintain software. (28:1)
>
> An environment that augments or automates all the activities comprising the software development cycle, including programming-in-the-large tasks such as configuration management, and programming-in-the-many tasks, such as team management.... an environment that supports large scale long term maintenance as well. (14:18)

The process, methods, and automation required to produce a software system. (9:38)

From the various descriptions encountered, key characteristics of a SEE can be identified. To acquire a better understanding of what a SEE is requires an examination of these characteristics, as well as the various types of SEEs and their architectures.

2.2 Characteristics of a SEE

In order for a SEE to provide adequate support for any software system engineering effort, it should exhibit the following characteristics:

1. Adaptability. A SEE should have the capability to adapt to changing project characteristics, behaviors and requirements (50:691; 60:39). A SEE should not be so rigid in its design as to limit its breadth of applications.

2. Level of automation. This pertains to the amount and types of automated assistance provided to the SEE user. A high degree of automation is desired for software engineering activities that, when done manually, are time consuming, tedious, or difficult to accomplish (50:691).

3. Level of integration. This refers to how well the different components of a SEE interact, and how this interaction appears to the user. When using the various SEE facilities, the user should notice little change in the look and feel among these facilities. This points to the requirement for a central data repository that all the SEE components would interact with (8:322; 49:39; 50:691; 60:39).

4. Reusability of internal components. SEE generated components of a software engineering effort should be available for reuse in the effort, or in other software engineering efforts. This characteristic also indicates the requirement for a central data repository (49:38).

5. Usability. The SEE should be user friendly and should include an easy to understand user interface. It should be easy for the intended users of a SEE to learn how to use it. A SEE should incorporate software engineering methodologies that are standard in practice, and should support the needs of project personnel (49:38; 50:690).

6. Utility. This concerns how the SEE can be utilized by the variety of users involved in the various software system life cycle activities. This includes programmers, designers, and managers. Utility also applies to software system life cycle activities. A SEE should support all life cycle activities (8:322; 49:37; 50:690).

7. Value. The cost of utilizing a SEE in time and money should offset the effort spent in learning and using the SEE. Increased savings in time and money should be evident when using a SEE as opposed to a manual effort (50:691).

2.3 Benefits of Establishing a SEE

The SEE characteristic of value, mentioned above, refers to the cost saving capabilities of a SEE when compared to a manual software system engineering effort. Besides these

benefits, a SEE provides other advantages that make it desirable to establish.

A SEE encourages good software engineering practice, and increases both the general quality of software systems and the productivity of software related personnel (50:691). A SEE also provides management and customer visibility into a project's progress and insights into the properties of the project's eventual results (9:35; 50:689).

In a study conducted in the early 1980s, Barry Boehm analyzed 63 software development projects with varying applications. Boehm found that the use of a SEE can reduce a development efforts cost by 28-41% (28:2; 33:56).

A 1980 report of the Government Accounting Office (GAO) indicates additional benefits. The GAO endorsed the use of software development automation and concluded that software tools (found in software engineering environments) can offer the following benefits (28:2):

1. Better management control of computer software development, operation, maintenance, and conversion.

2. Lower costs for operation, maintenance, and conversion.

3. A feasible means of inspecting both contractor developed and in-house-developed computer software for such quality indications as conformance to standards and thoroughness of testing.

In summary, the primary benefits of establishing a SEE are that it provides a reduction in cost, development time, and effort over the system life cycle, and an increase in

productivity of personnel and quality of the software system produced.

2.4 SEE Types

Before examining some of the various SEE types it is important to point out that various authors differ in their views of the correlation, if any, between a SEE, a software development environment and a programming environment. Steubing, for example, contends that the latter two environments are concerned primarily with the code and test activity of a software system life cycle and deal specifically with software. He states that a SEE is different in that it provides support over the entire life cycle, including aspects other than software (61:388). Others support a different view, stating that a software development environment and a SEE are the same environment (14:18). The different categories of environments presented in this thesis were determined by various authors with differing opinions on how software environments should be classified.

There are few, if any, existing SEEs that support all software engineering activities in a software system life cycle (26:14). Instead, since activities that occur in earlier stages of the life cycle are different from those that occur later, a SEE tends to concentrate on one stage or the other (28:5). This results in various environment types that can be grouped into six basic categories: framing environments, method based environments, structure oriented

environments, programming environments, toolkit environments, and general environments (3:2; 14:18; 28:5). Any particular environment may fall into a number of the presented categories.

Framing Environments. This type of environment is concerned with the earlier stages of a software system life cycle. The name framing is derived from the fact that the system is concerned with the stage of the life cycle where requirements and design frame the system (14:18; 28:6).

Method Based Environments. Method based environments support a specific method for developing software. They support particular development methods and the management of the development process (3:4). Examples of method based environments are Excelerator, CASE 2000, and Cadre Teamwork.

Structure Oriented Environments. This type of environment is "constructed around a central structure editor for manipulating structures such as abstract syntax trees" (3:3). Structure oriented environments are language dependent, support direct manipulation and multiple views of a program structure, and support only the coding phase of the software system life cycle.

Programming Environments. Programming environments, also known as language centered environments, are built around a specific language, and as such, provide features such as tool sets that are oriented toward that language (3:3; 14:18;

28:6). This type of environment is primarily concerned with the latter stages of the life cycle. Example of this type of environment are the Ada Programming Support Environment (APSE) (34:16; 15:1), and the Rational environment for Ada (3:3).

Toolkit Environments. This environment provides a collection of tools that support programming-in-the-large tasks such as configuration management and version control (14:18). It is similar to a programming environment in that it supports the latter stages of the software development process (3:4), but it differs in that it is programming language independent.

General Environments. This type of environment supports all phases of the life cycle, and is independent of a specific programming language. General environments support programming-in-the-large and provide both basic tools, such as compilers and editors, as well as CASE tools (14:18; 28:7). UNIX and VMS are examples of this type of environment. An Integrated Programming Support Environment (IPSE) is a general environment (35:52; 42:1) that also supports the management of the software process (3:2).

2.5 SEE Architectures

A software environment architecture refers to the components of that system, their interrelationships, and interactions. An architecture can display the characteristics of a combination of four basic types of architectures:

control-centric, data-centric, network, and virtual machine (50:691). Most environments will exhibit a mixture of these architecture attributes.

Control-centric architectures organize components in terms of flow of control among the components, with the internal supervising subsystem at the heart of the environment. The core of a data-centric architecture is the data repository around which components are organized in terms of data flow relations with the repository. Network architectures utilize message communication between a network of processes. Finally, virtual machine architectures, such as an APSE, organize components of the environment into implementation support layers, with each lower level supporting the implementation of the higher ones (34:17; 15:1). This thesis will concentrate on this type of architecture.

A virtual machine architecture is composed of four layers: the hardware and native operating system layer, the environment support layer, the tool/capability layer, and the environment adaptation and project user support layer (34:14; 50:692).

The hardware and native operating system layer is the lowest layer of the architecture and is comprised of the underlying hardware and native operating system.

The environment support layer provides a set of commonly used facilities. It is composed of the virtual operating

system, an object management system, a user interface management system, and an environment management system.

The tool/capability layer is composed of functional tools and capabilities, tool building tools, and integration mechanisms. These components provide the environment's functionality.

The environment adaption and project user support layer is the highest layer in the architecture. It consists of adaption support mechanisms and project user support capabilities, and is oriented toward the needs of environment adapters and project users. An example of this architecture is given in Figure 2.1.

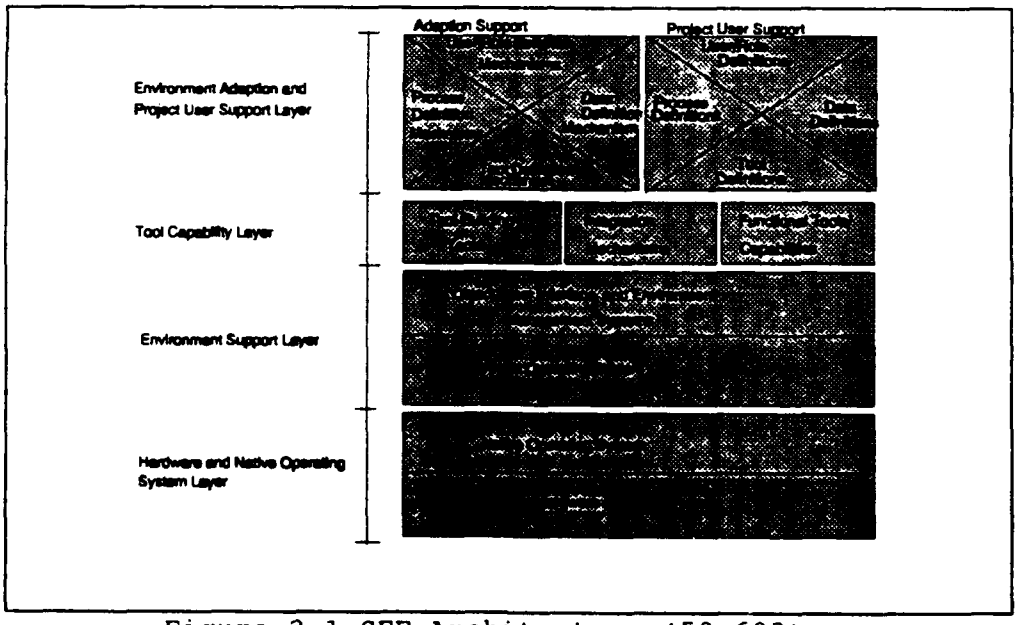


Figure 2.1 SEE Architecture (50:693)

One example of a SEE is an APSE. An APSE is a programming language specific SEE, as indicated by the title. The purpose of an APSE is "to support the development and maintenance of Ada applications software throughout its life cycle, with particular emphasis on software for embedded computer applications." (15:1)

An APSE architecture essentially provides the same services as the virtual architecture presented by Penedo, but the layers are labeled differently. The three layers are the kernel Ada program support environment (KAPSE), the minimal Ada program support environment (MAPSE), and the Ada program support environment (APSE)(8:323; 34:17; 15:1). An example of this "onion skin" architecture is given in Figure 2.2

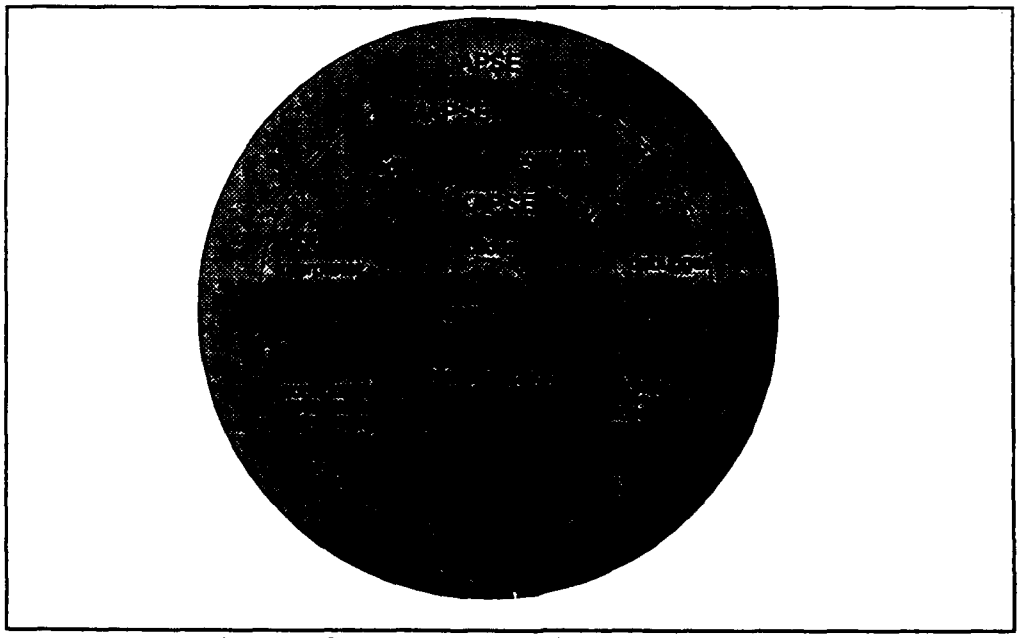


Figure 2.2 APSE Architecture (15:1)

The KAPSE contains all system and implementation dependencies, and is comparable to the hardware and native operating system layer of Figure 2.1. The MAPSE provides a minimal set of tools necessary for the development and support of Ada programs, and is roughly comparable to the tool/capability layer of the virtual architecture. The standard interface in Figure 2.2 is roughly comparable to the environment support layer which simplifies the construction of the components in the tool layer of the virtual architecture by providing a set of commonly needed facilities. The APSE outer layer of Figure 2.2 is roughly comparable to the environment adaption and project user support layer. The APSE provides the capabilities for process management and project specific design and management.

2.6 Methodologies for Evaluating a SEE

Numerous methodologies can be used when evaluating a SEE (17:31; 64:199). Weiderman and others define a methodology for evaluating environments, for the purpose of "adding a degree of rigor and standardization to the process of evaluating environments" (64:199). Without an evaluation methodology, most environment evaluations are ad hoc at best. They stress that any methodology should be based on user activities, independent of any actual environment, experimentally based, should test a core of functionality, and should be evolutionary and extensive. The 6 phases of their proposed methodology (64:201) are as follows:

Phase 1 - Identify and classify software development activities.

Phase 2 - Establish evaluation criteria.

Phase 3 - Develop generic experiments.

Phase 4 - Develop environment specific experiments.

Phase 5 - Execute environment specific experiments.

Phase 6 - Analyze the results.

The scope of this thesis prohibits a full implementation of this methodology. Phase 1 activities, however, can be identified as a way to better understand the SEE and the CASE tools that may be required as part of that environment.

2.7 CASE Definition

As previously discussed, one of the characteristics that should be exhibited by a SEE is that it automates the activities of a software system life cycle. Automation can be accomplished through the use of various software tools known as CASE tools. CASE is defined as follows:

> CASE can be viewed as an environment that supports the software engineering process. (47:1102)

> CASE is the development and effective use of appropriate management approaches, systematic procedures and methods, and automated tools that permit teams of software engineers to produce software that 1) meets business and systems requirements; 2) is completed with a predictable schedule; 3) is available within budget guidelines; 4) allows for easy maintenance and enhancement. (57:376)

> (CASE) technology is the industrialization of software development. It is an integrating technology which will pull together and focus the many methods, techniques, and tools software developers are using or will be using in the future. (44:373)

Many authors describe CASE as the automation of the entire software life cycle with a set of integrated tools (22:190; 46:33). Sharon further describes CASE as a term that applies to all methods, procedures, techniques and tools used to engineer software (57:376).

A comparison of the definitions of a SEE and CASE environments indicates the two are very similar, if not synonymous. In particular, three aspects of an environment are common in most of these definitions: a software process, a method, and automated tools. The remainder of the chapter will deal mostly with the third aspect, automated tools, in particular, CASE tools.

2.8 Integrated CASE Environments

Before the discussion of the various types of CASE tools can begin, a highly desirable feature of a CASE or SEE environment needs to be considered. An environment, such as the CASE environment presented in Figure 2.3, exists in most software development organizations in varying forms.

What is usually lacking among tools used in this type of environment as well as in the environment itself is integration (57:377). Penedo and Riddle's virtual architecture, Figure 2.1, compensates for this deficiency by incorporating integration mechanisms in the tool capability layer. When new CASE tools are being evaluated, consideration has to be given to the efforts on current manual techniques and existing tools to determine the level of integration that

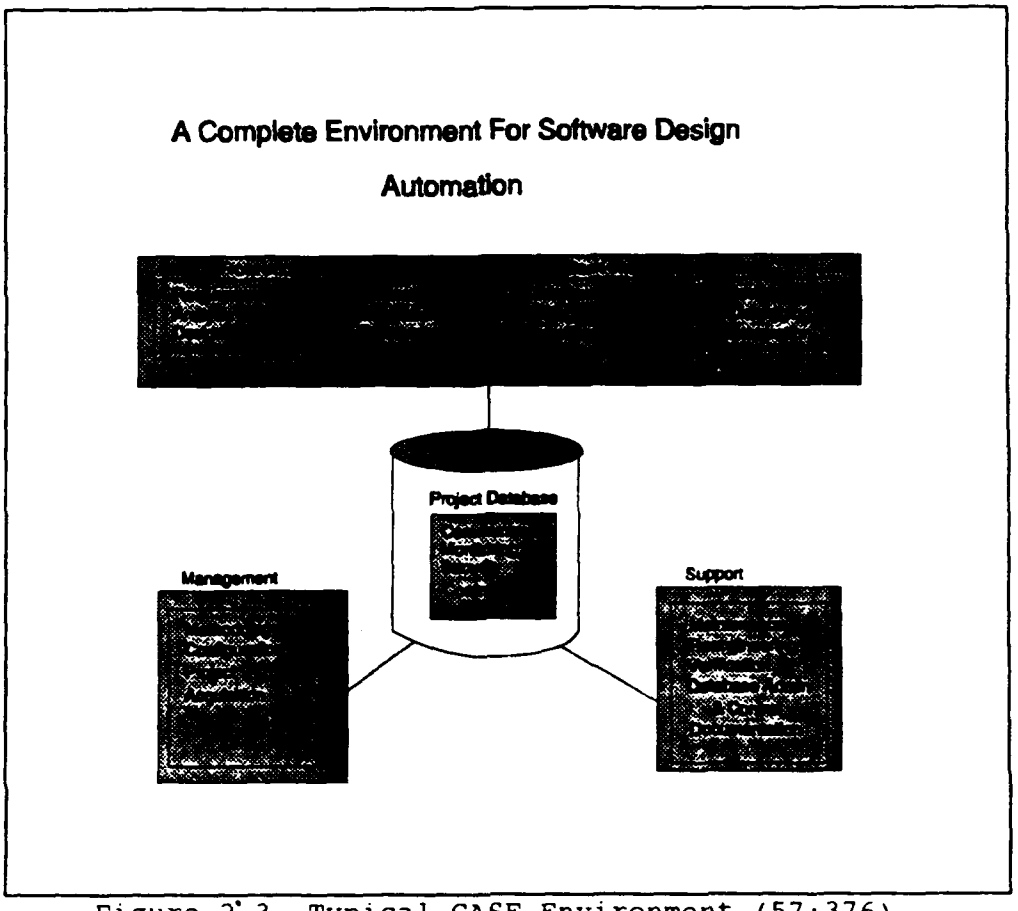


Figure 2.3 Typical CASE Environment (57:376)

exists. Consideration must also be given as to how well the new tools will be integrated. Figure 2.4 displays the dimensions of software engineering which evolve into integrated CASE environments.

Forte points out that:

> while individual CASE tools each contribute to improved quality and productivity in software engineering, the promise of CASE lies in the potential to integrate many tools into an integrated environment. (18:5)

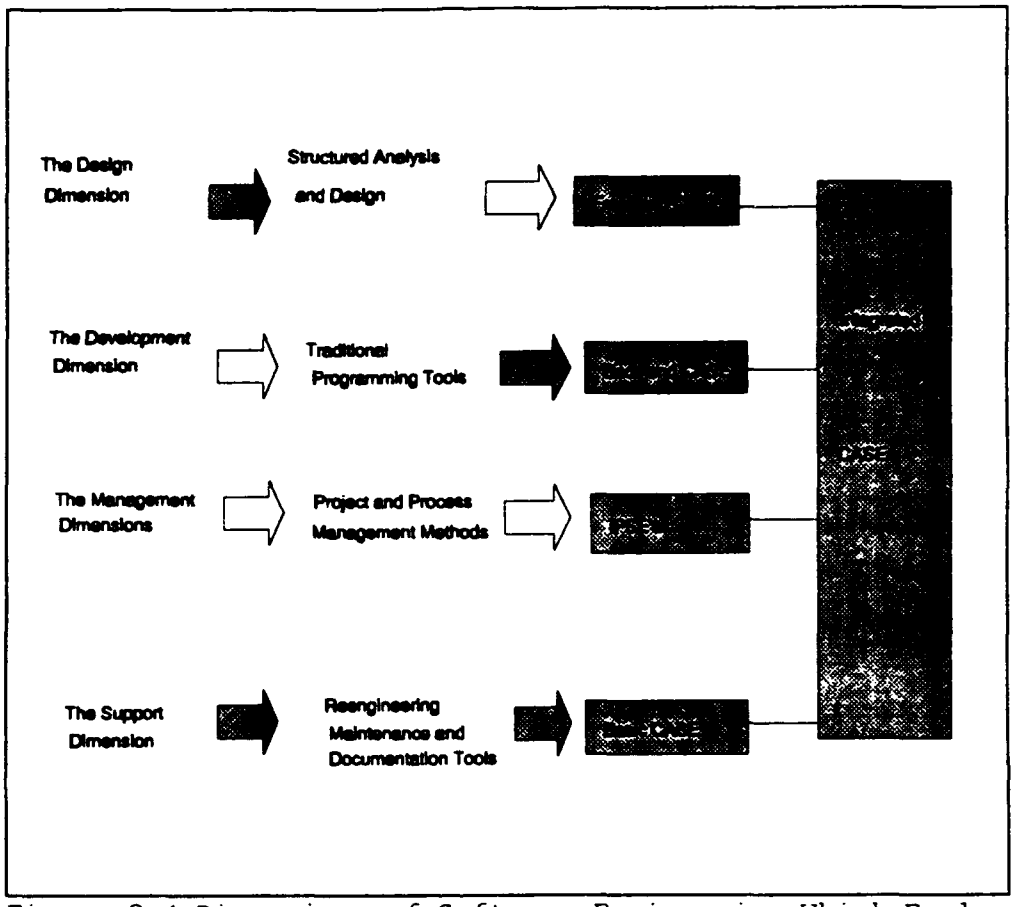


Figure 2.4 Dimensions of Software Engineering Which Evolve to an ICASE Environment (57:377)

Some benefits of integration of CASE toolkits (18:2) are:

1. Design information is preserved from activity to activity in the software systems life cycle, eliminating redundant input.

2. Bookkeeping tasks, such as those associated with software design, construction, and documentation, can be automated.

3. The ability to plan, monitor, and control the software development process is improved.

4. The reconciliation and combination of similar design representations and methodologies is encouraged.

2.9 Types and Categories of CASE Tools

CASE tools are automated software tools that "support the implementation of both procedures and methods and tend to reduce the dog work associated with manual procedures" (53:1). Although "the ultimate goal of CASE tools is to automate software development from design to coding" there is no one CASE tool that "does it all" as far as providing automated support across the software systems life cycle (62:184). Instead, there are a number of types and categories under which a particular CASE tool may be classified.

Software Tool Categories. As mentioned previously, a variety of software tools are utilized as part of a SEE. These software tools can be grouped into 3 categories: Project management tools, system design and verification tools (technical tools), and support tools (11:544; 31:1).

Project Management Tools. These management tools include tools which allow the project manager the capability to track and project the progress of a project. Examples of these tools are spreadsheets, project cost, schedule and tracking estimates (11:544), and data modeling and life cycle tools (58:23).

System Design and Verification Tools. These technical tools deal with the analysis, design,

implementation, verification, maintenance, and documentable efforts involved in the software engineering effort (11:544; 58:22).

Support Tools. These types of tools can be further categorized into two types, system development support tools and system performance support tools.

System development support tools include automated support tools such as data base management systems, precompilers, syntax checkers, and others (11:545).

System performance support tools include software and hardware tools that assist in "tuning system hardware, software, and communications to meet the requirements of new systems development" (11:545).

CASE Tool Categories. In the course of this research effort, many categorizations for CASE were presented by various authors. One such categorization groups CASE tools into 3 classes: Front end, back end, or reengineering CASE tools (7:22). Front end CASE tools provide support for the analysis and design phase of the software systems life cycle. Back end tools provide support for the construction phases of the life cycle, the development and implementation phases. Reengineering tools support the maintenance phase of the software life cycle.

Another categorization of CASE tools is lower CASE and upper CASE tools (4:73; 12:380; 52:10). Coallier describes lower CASE tools as "programming oriented text based-tools,"

similar to the description of back end tools (12:380). His description of upper CASE tools as "tools that address the needs of software systems analysis and design" is similar to the description of front end tools (12:381).

Yet another categorization of CASE tools describes them as either vertical or horizontal (3:2; 18:6). Vertical CASE tools are those which support the software engineer from the requirements analysis phase to the implementation phase of the software system life cycle. These tools are associated with a specific phase of the software development life cycle. Horizontal tools are software system management tools used for documentation, configuration management, project management, communicating and system building. These tools are associated with different phases of the software development process.

2.10 Toolkit Vs. Workbench

Baram and Steinberg make a distinction between the terms "toolkit" and "workbench." They define a toolkit as a collection of CASE tools that support a particular phase of the software systems life cycle. Workbenches are defined as a complete set of tools that support the entire life cycle and are integrated so that specifications are passed from one phase of the life cycle to the next. Based on their definitions, they contend that few complete workbenches exist (4:73).

2.11 Benefits of CASE Tools

The benefits of implementing CASE tools as opposed to

using manual methods to accomplish software engineering efforts are numerous. Some of the more obvious benefits are increased productivity (7:23; 33:56; 32:220; 36:105; 45:89; 65:349), increased product quality (16:1; 32:220; 36:105; 45:89; 65:349), decreased project costs (16:5; 22:190; 36:105), and a decrease in the difficulty of maintaining a generated project (7:24; 58:23; 65:349). These are similar to the kinds of benefits experienced as a result of establishing a SEE, which is expected since CASE tools are an aspect of a SEE. Many of a SEE's benefits can be attributed to the automated tool aspect.

Other benefits of incorporating CASE tools are:

> Training and enforcement in the development methodologies used, interactive graphics that support the diagrams of analysis and design, single-entry documentation book keeping that enables redundant specifications to be generated from nonredundant dictionary contents, and reminders and consistency checks (65:346).

> Efficient allocation of staff, increased efficiency, and minimal errors. (38:28)

Numerous studies have been conducted which confirm these benefits. Barry Boehm's landmark survey of productivity mentioned earlier in this thesis is an example of one such study (28:2; 33:56).

Lempp and Lauber conducted a survey concerning the Ada analysis, design and coding CASE environment, EPOS, studied the impact of CASE environments on productivity, project costs, software quality, and the impact of people working with CASE tools (37:105). The results indicated an overall increase in productivity, a 9% overall cost reduction, and

69.2% fewer specification and design errors. The study also found that user acceptance of CASE tools increased over time. The survey results indicated moderate savings throughout the development phase, considerable improvements in software quality, with revolutionary cost savings in maintenance being achieved as a result of increased error detection in the earlier phases of a project.

A study conducted by Charles Necco concerning the use of CASE tools in computer based information systems found that of the 15 organizations surveyed, 47% found a significant improvement in productivity when CASE tools were employed, while the remaining 53% noticed moderate improvement. In addition, 60% of the surveyed organizations noted significant increases in product quality, while 40% noticed moderate increases in this area (45:7).

Baram and Steinberg cite a study by M. Loh and R. Nelson in which the biggest productivity gains of implementing CASE tools were as a result of the automation of tasks such as diagram drawing (4:73). Manual efforts in this area prove both tedious and time consuming. A major problem of system developers has been the "administrative burden of creating and maintaining a large set of drawings" (63:109).

Similar results were found in a study by R. Norman and J. Nunamaker. Ninety-one subjects ranked data flow diagrams (DFDs) as the number one function desired in a CASE tool. The reason cited was that the manual creation of DFDs was viewed as extremely time consuming (47:1104).

The aforementioned benefits make a strong case for the implementation of CASE tools in a software engineering organization.

2.12 Potential Pitfalls of CASE Tools

Although numerous benefits exist as a result of implementing CASE tools (as identified in the previous section), there are also potential pitfalls that CASE users should be aware of. A combination of these pitfalls and misconceptions concerning CASE tools can result in a loss in productivity rather than the anticipated gains that should be realized when implementing CASE tools (10:120). The reasons given by Chikofsky for this productivity loss are:

1. Organizations fail to comprehend the resources required for successfully introducing a tool.

2. Tools are viewed as a means of saving a project that is in crisis. In other words, tools are not viewed as simply tools, but as problem solvers.

3. Organizations use the same tool in the same way for all projects, failing to customize tools to the intended use (if possible) or obtaining new tools if required.

4. Blaming the tool for anything that goes wrong with a project.

5. Using the wrong tool for the job.

This last reason, selecting a tool that does not meet the user's needs, is a common pitfall (39:17-1; 40:40). The principal causes for this situation could be that tools are

selected before a SEE's software development process and methods are established, or without an understanding of an existing process or methods (this is discussed in more detail later in the chapter). Other reasons may be the tool's capabilities are incomplete or inadequate, the choice of a tool is made prematurely, there are problems with the tool's portability, the tool is too complex, or the tool is not robust (39:17-1).

Clearly, a sound understanding of what a CASE tool's capabilities are, which tools are required for a project, and what they can provide the user, should exist to prevent the misuse of these tools. Useful tools should be introduced into the environment on some orderly basis (23:3).

Wilson identifies what he considers to be inherent limitations of CASE tools (65:346). Some of these limitations are:

> Lack of integration makes tool selection a time consuming and tedious process and implementation difficult to effect; integration becomes the responsibility of the purchaser.
>
> Constraints on the methodologies supported may mean that the tool does not provide the flexibility for the developers to apply the techniques with which they are familiar, or that are appropriate for a particular application.
>
> Advanced graphic capabilities are often offset by poor documentation production capabilities, which inhibit the availability of well designed hard copy specifications for user review.

Gibson states that many of the problems that occur with CASE tools are a result of human misconception (21:12). He identifies his "Baker's Dozen" of major misconceptions

concerning CASE and integrated CASE (ICASE). These misconceptions are presented in Table 2.1.

Table 2.1 Major Misconceptions Concerning CASE and ICASE (21:12)

1. CASE is a methodology or replaces existing methodologies or techniques.
2. CASE is a monolithic concept.
3. CASE is an extension of or replacement for fourth generation languages.
4. All CASE systems have the same structural frameworks and outputs.
5. Using CASE will make you a better strategic manager or system professional.
6. CASE obviates the need for discipline and close management of application development.
7. The most important output of CASE is applications development software.
8. Productivity gains with CASE are immediately evident.
9. Use of CASE insures consistency of output.
10. CASE will eliminate the DP applications development backlog.
11. CASE will eliminate the systems analyst.
12. There is no difference between ICASE and CASE.
13. Reengineering and reverse engineering are synonymous in the CASE environments.

The potential for problems in the implementation of CASE tools is evident. Many of these problems can be avoided by insuring that a comprehensive evaluation of the SEE and of the CASE tools under consideration for use is performed.

2.13 The Software Process

Prior to evaluating and selecting a CASE tool to automate the activities of a software systems life cycle, a "systematic set of software development processes and methods" (58:23) necessary for successful software engineering, must be defined (29:9; 58:23).

Humphrey defines the software process as "the set of tools, methods, and practices we use to produce a software product" (30:3). Smith identifies the software process as one of the three aspects of software engineering, similar to Charette's description of a SEE as well as Sharon's description of CASE given earlier. These aspects are defined as follows:

> Processes - Sequences of phases, tasks, and activities required to develop software. One part of the software process is the sequence of software life cycle steps as they are defined in a particular organization.
>
> Methods - Techniques used to perform specific defined tasks and activities. Several methods may support each task or activity in the software development process. For example, the task of constructing a data model may be performed using the Chen, Merise, or Bachman techniques.
>
> Tools - Automated support for processes and methods. (58:23)

The process and methods of a SEE must be established prior to tool selection. Otherwise, the process or method might be

modified to fit the tool chosen. According to Smith, "far too many organizations have purchased a tool, then tried to force-fit its own culture and methodology to the demands of the tool" (58:24). Without a process and methods, an organization will be unable to take full advantage of a tool's capabilities.

2.14 CASE Tool Criteria

Various ways exist to group criteria used to evaluate CASE tools. Baram and Steinberg group selection criteria into ten categories: Learning Curve, User Interface, Data Dictionary, Analysis, Reports, Graphics/Diagramming, Interfacing to Other Systems, Text/Documentation, Project Management, and Prototyping (4:74-80). Each of these criteria is representative of a lower level group of criteria.

Lawlis specifies two sets of criteria, a list of thirteen top level absolute criteria which, like Baram and Steinberg's list, can be further decomposed into lists of detailed relative criteria (34:343-357). Regardless of the way in which the criteria are grouped, it is important to point out that not all criteria may apply. The chosen criteria should be based upon the SEE in which the CASE tools will be integrated. The evaluator must choose the criteria that apply to their environment and the software process used.

As an example of criteria that might be used to evaluate a CASE tool, this research considers the criteria grouping proposed by Firth and others. They group the criteria used to

evaluate CASE tools according to the "aspects of a tool's acquisition, support, and performance they address" (17:19). These aspects are ease of use, power, robustness, functionality, ease of insertion, and quality of commercial support. Although this is a comprehensive list of criteria, it is by no means complete or all inclusive. For example, criteria such as reusability, survivability, and integrity (34:344) are omitted.

Ease of Use. Ease of use refers to the "user friendliness" of a tool. Aspects of ease of use are the human/system interface, helpfulness, error handling, and tailorability.

The Human/System Interface. Features of the human/system interface that should be considered when evaluating a CASE tool include windows for multiple views, an easy to use menu system, multiple user capabilities, mouse capabilities, and display features such as color and reverse video (4:74; 17:21; 20:65; 53:6; 62:127; 66:61).

Helpfulness. Features of helpfulness to consider are the availability of an on line help facility and a tutorial, multiple modes of operation based on the user's skill level, and useful graphics features such as icons, shape and texture (4:74; 17:20; 20:65; 22:192; 34:355; 53:6; 62:128; 66:61).

Error Handling. Features of error handling are error recovery capabilities, error prevention capabilities, automatic back up capabilities, error detection and correction, and inconsistency detection (4:75; 17:20; 53:6).

Tailorability. Features of tailorability are the capability to shut off unwanted features, the capability to reformat input and output, and the availability of vendor assistance in tailoring the tool to an organization's needs (17:19; 34:355).

Power. Power is concerned with the various performance features of the tool. The different aspects of power are structural modifiability, leverage, state and performance.

Structural Modifiability. Features of this aspect are the capability to operate on view objects (i.e. DFDs) at different levels of abstraction or detail, the capability to modify collections of objects, the availability of modification features such as zoom, insert, delete, or modify, and the capability to modify view objects freehand (4:77; 17:21; 20:65; 22:192; 53:6).

Leverage. Features of leverage to consider are the capability for users to add macros, and the application of commands systematically to the entire collection of view objects (4:75; 17:22).

State. This feature refers to a CASE tool's capability to remember previous and current session usage, providing the user the capability to save and restore view objects (17:23).

Performance. Performance refers to a tool's capability to function efficiently and be responsive to the user. Considerations of this feature are the capability of the tool to support multiple users, tool response time, command execution time, maximum workload, and the capability to handle the size of a required task (17:23; 22:193).

Robustness. Robustness of a CASE tool refers to the reliability and consistency of the tool. Aspects of robustness are consistency, adaptability, and tool maintainability.

Consistency. This feature deals with the consistency of tool operation, such as the capability to store output and then retrieve and access that output (17:24; 22:193; 34:361).

Adaptability. Adaptability refers to a CASE tool's ability to evolve over time due to changing requirements or a changing environment, or for tool enhancements (17:24).

Tool Maintainability. This feature refers to a tool's capability to be examined and possibly repaired by the

user in the event that tool flaws or bugs occur (17:25; 34:350).

Functionality. Functionality deals with a tool's methodology support and correctness.

Methodology Support. This feature is concerned with a tool's capability to support one or more software engineering methodologies, as well as supporting all aspects of a methodology. Methodology support is also concerned with the ability of a tool to integrate additional methodologies and to adapt to in house methodologies (13:29; 17:25; 20:65; 22:192; 53:6; 62:127).

Correctness. This feature refers to a tool's ability to operate correctly and produce correct output (17:26; 34:362).

Ease of Insertion. Ease of insertion refers to how easily a tool is incorporated into an environment based on the learning curve of the tool and the SEE itself.

Learning Curve. This aspect refers to the complexity of the tool and the ability and background of the user. Consideration should be given to the time required to learn a tool, novice vs. expert capabilities, and command consistency (4:74; 17:26; 53:6; 58:27; 62:127).

Integration. This aspect refers to the ability to integrate the tool into the SEE. Consideration should be

given to the similarities between the tool under consideration and existing tools in the SEE, the tool's compatibility with the SEE hardware and operating system, the ease of installation, use of a common database, and how well the tool integrates with other tools in the SEE (13:29; 17:27; 20:65; 22:193; 60:39; 66:61).

Quality of Support. Quality of support refers to the history of the tool and vendor history and support.

Vendor History. This feature concerns the vendor's past performance concerning user support, as well as the vendor's reputation (17:28; 34:356; 53:6; 58:27; 62:128).

Tool History. This aspect concerns the use of the tool under consideration at other organizations and evaluation of its performance at those organizations (17:28; 53:6; 62:128).

Vendor Responsibilities. This aspect concerns vendor responsibilities such as tool installation support, training on tool usage, types and amount of documentation provided, maintenance responsibilities of the vendor, and responsible feedback from the vendor to the user's questions, possibly by use of a hotline (11:550; 13:29; 17:29; 34:356; 40:44; 53:6; 58:27; 62:128).

Legal Considerations. This aspect deals with the purchasing, licensing, or rental agreements concerning the

tool, as well as the cost of the tool , explicit contract agreements, refund periods, source code accessibility rights, and availability of site licenses (11:551; 17:28; 34:356; 53:6; 58:27; 62:128).

2.15 A Process for Evaluating CASE Tools

Defining and applying criteria for the evaluation of CASE tools is just one step in selecting and applying CASE tools. This is a four step process that involves a needs analysis (17:31; 25:371; 54:357), an environment analysis (17:31; 67:42), the development of a CASE tool candidate list (17:32; 62:125; 67:43), and the application and selection of a CASE tool, toolset, or work bench (17:33; 62:125; 54:34).

Needs Analysis. In this step of the assessment process, the purpose for which a tool would be needed is determined. Before choosing a tool "the development process to which they are applied must be established" (25:371). The tool must make a contribution to the overall task of an organization. "It must contribute to a process controlled by a method" (17:31). This step begins by including all personnel affected by the incorporation of CASE tools into the environment in the analysis process (54:357). Questions to be considered at this phase are (17:31; 54:357):

1. What model of software development is used by the organization?

2. What major tasks are required by the model?

3. Which tasks could be better performed with automated tools?

4. Which tasks are lacking in adequate tool support?

5. What are the perceived benefits to be obtained from specific new tools?

To be successful in the implementation of CASE requires understanding and cooperation in the early phases of the process. The organization should understand the overall software development process before deciding to acquire tools. This involves getting the full cooperation of everyone involved (54:357).

Environment Analysis. This step involves the analysis of the organizational environment in which the tool will be used. Environment constraints such as economics, time, skill level of personnel, etc., need to be considered to determine if a tool can be used successfully in the environment (17:31).

Develop a Candidate List. This step entails the compilation of a list of potential CASE tool candidates. This can be accomplished by surveying the software market to determine available tools (62:124). Other means of identifying candidate tools include contacting vendors for information (67:43), attending trade shows, obtaining trade publications, and surveying technical journals (17:32; 54:361).

Applying Criteria and Selecting a CASE Tool. This step in the tool selection process involves five phases: establishing criteria, determining test cases, executing test cases, analyzing test results, and choosing a tool.

Establishing Criteria. A sampling of evaluation criteria was presented earlier in this chapter. Each organization needs to determine what criteria in the provided list should be applied, as well as additional criteria that must be augmented as a result of the organizations needs (17:33).

Determining Test Cases. Once a criterion is established, experiments need to be developed that fully test that criterion (17:33). Testers and evaluators should expect to spend time learning the tool before applying functional tests (67:44).

Execute Test Cases. It is important in this step not to rely too heavily on the product literature and documentation as a source for testing criteria. The tests conducted should be by hands on use of the tool (17:33). If early results show inadequacies in the tool, further testing may not be required.

Analyzing the Results. In this phase, the results of all the tests are analyzed. The results of the tests for each CASE tool under consideration are first examined individually, then collectively to compare the CASE tools. The

analysis should "determine how well the tool satisfies each of the criteria" (17:34).

Selecting The Tool. The final phase of the entire process is the selection of the CASE tool that best suits the needs of the organization (17:34; 54:362; 67:44). It is important to remember that a tool may not be found that is a perfect match for the organization's needs. The best that can be hoped for, short of in-house development of a tool, is to select the one that best matches the organization's needs.

2.16 Summary

An established SEE is highly desirable for any software systems development organization. A SEE can be classified based on the support given to various stages of the software systems life cycle, and can be further described by its architecture. Although each SEE is unique, there are three common factors that every SEE is composed of: a software process model, at least one method, and automated tools. An understanding of the process and method(s) is required before the appropriate automated tools can be evaluated and selected for integration into the SEE.

In chapter three, the SEE at AFIT will be examined and defined. The criteria for evaluating the CASE tools under consideration, as well as the evaluations used, will be presented. Finally, a heuristic for evaluating the CASE tools will be introduced.

III The Software Engineering Environment

This chapter addresses research question one, "How can a software engineering environment at AFIT be defined?", and research question two, "What criteria should be used to evaluate the CASE tools under consideration?" The chapter begins with a description of the SEE at AFIT. The type of SEE as well as the architecture are presented. The categories of the CASE tools being examined are then defined. Next, the criteria to evaluate the CASE tools under consideration are established. Questionnaires for use by CASE tool evaluators are discussed, as well as numeric weight assignments for the criteria. The approach taken to evaluate the CASE tool sets is then explained. Test cases to be used during the evaluation process are discussed. Finally, a methodology for software evaluation and selection is presented.

3.1 The SEE at AFIT

The type of SEE that currently exists at AFIT can be described as a general environment. As discussed in chapter two, this type of environment supports all phases of the software systems life cycle, independent of a specific programming language (although Ada is the predominant language used). This general environment at AFIT can be further viewed as a distributed environment composed of several separate environments: a UNIX environment, a VMS environment, a Rational environment, and a PC environment. The VMS and PC

environments can be classified as general environments. The UNIX environment at AFIT lacks the tools to support the earlier stages of the software development life cycle, primarily supporting the implementation and maintenance stages, and can be classified as a toolkit environment. The Rational environment can be classified as both an APSE and a programming environment since it supports software development using Ada exclusively. These four environments communicate via the AFITNET network. A general architectural view of this distributed environment is presented in Figure 3.1.

The architecture of the VMS, UNIX, and PC environments can be viewed as a three layered architecture consisting of a hardware layer, an operating system (OS) layer which includes both native and virtual operating systems, depending on the machine being considered, and a tool support layer. An environment adaptor and project user support layer, such as the one presented in Penedo and Riddle's architecture example in chapter two, is not considered for these environments. In a software development organization with an established software process model, this layer would be utilized by both environment adapters who select and compose the components found in the various architecture layers to provide a project specific environment, and project users who build application systems using the project specific environments provided by the environment adapters (50:692). This layer is considered,

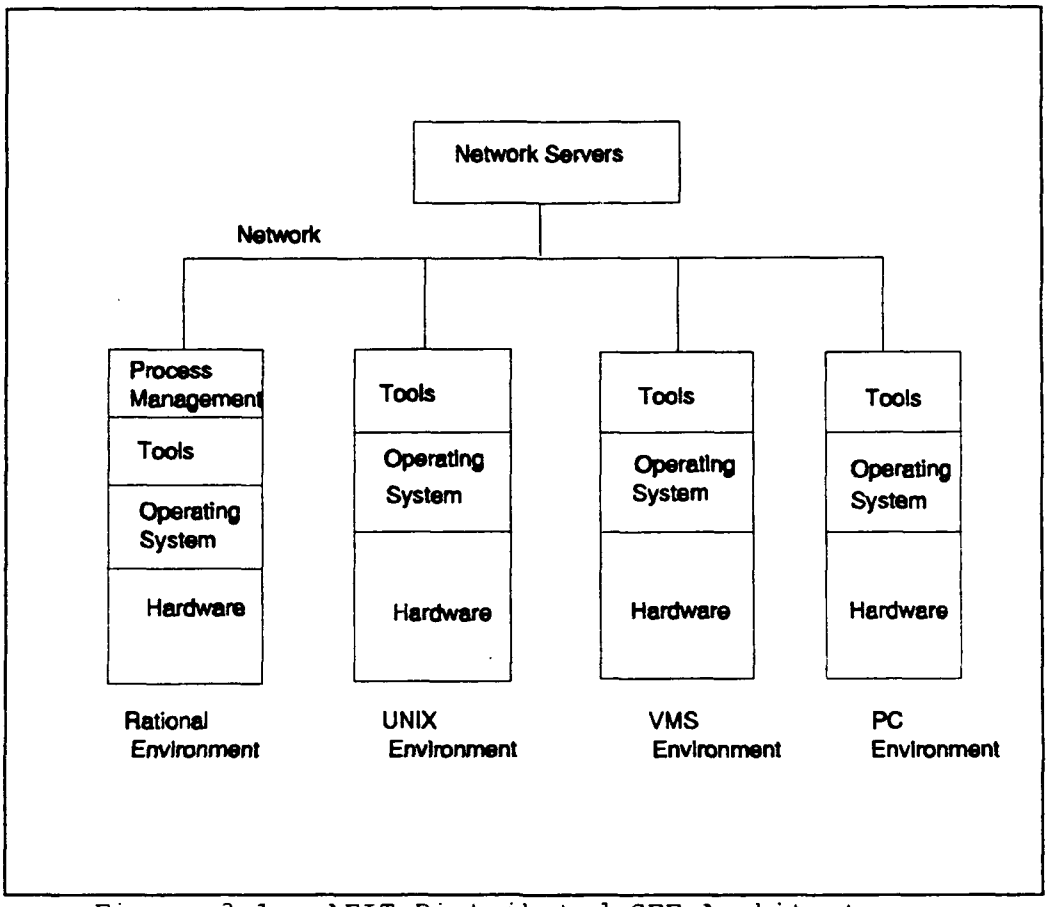


Figure 3.1 - AFIT Distributed SEE Architecture

however, for the Rational environment, since the capability to manage the process exists.

The VMS Environment. The VMS architectural hardware layer consists of a DEC VAX 6420, a DEC VAX 8550, a VAX 8650, a DEC 11/785, ten VAX Station IIIs, a VAX Station II, two micro VAX IIs, and two micro VAX IIIs.

The OS layer consists of the VMS OS, which serves as both a native OS and virtual OS for most of the machines, and

handles object, interface, and environment management.

The tool layer is composed of a set of tools necessary to support the software development life cycle. Tools that support requirement analysis and design are DECdesign, VAXset with Program Design Facility (PDF), and Excelerator. Implementation tools include: compilers and editors to support the Ada, BASIC, BLISS, C, FORTRAN, LISP, and Pascal programming languages; VAX Debug for debugging; VAX LSE/SCA for language sensitive editing and source code analysis; VAX DEC/Code Management System (CMS) for code and configuration management; VAX DEC/Module Management System (MMS) for system building; VAX DEC/Test Manager (DTM) and VAX Performance and Coverage Analyzer (PCA) for integration and testing; DECwrite and VAX DOCUMENT for documentation; VAX Notes and Mail for communications; DECwindows, DEC GKS, and DEC PHIGS for user interface tools; Rdb/VMS with VAX SQL and VAX CDD for information management.

The UNIX Environment. The UNIX architectural hardware and OS layers consist of: an Elxsi 6420 which executes Embos, Elxsi's native OS and UNIX 4.3 bsd virtual OS; a VAX 11/785 which executes UNIX 4.3 bsd virtual OS; an Elxsi 6400 which executes Embos and UNIX 4.3 bsd; Three Counterpoint 22e machines that execute AT&T System V OS; Sun work stations which execute Sun OS 4.1.1 (AT&T System V with bsd extensions).

The tool layer of the UNIX system is composed mainly of tools that support the implementation stage of the software development life cycle. Programming languages supported are Assembler, Ada, C, FORTRAN and Pascal. SLAMII is available as a simulation language and is supported by TESS which can be used to graphically prototype SLAMII simulations. Language editors include the UNIX vi editor and EMACS. Tools for debugging, code reformatting, file transfer, data modeling, and configuration management are available as well as a window system for interfacing, mail and write tools for communications, and Word Perfect, Publisher, and LaTex for documentation.

The PC Environment. The PC hardware and OS layers of the AFIT SEE architecture is composed of 386 and 286 based PCs which execute the Disk Operating System. The tools available in the tool layer which support software development include: data modeling tools such as DBASEIII, Paradox, and INGRES; system design and verification tools such as POSE, ObjectMaker, Design Aid, NASTEC 2000, and Excelerator; implementation tools that support the Ada, BASIC, C, PROLOG, Pascal, FORTRAN, and LISP programming languages; file transfer tools including Kermit and File Transfer Protocol; tools for documentation, including LaTex, Word Perfect, WordStar, TED, EDT, and MicroEMACS.

Not all of these tools are visible to every AFITNET user. Some of the tools, such as Excelerator and NASTEC 2000, are

available on a project or user need basis. Access to these tools are controlled by the systems administrator.

The Rational Environment. The hardware layer of the Rational environment is composed of a Rational R1000 series 400 single cabinet machine. There is an operating system that performs environment management functions and is referred to by revision number, currently D-2. Since Rational is a self contained environment, no specific tools are designated that perform functions which support the different stages of the software development life cycle. Instead, software development support is considered to be provided by the environment (55).

The Rational environment is a totally integrated, interactive, knowledge based environment for Ada. A complete Rational environment supports Ada software development through system design, as well as system reengineering, and project organization facilities for editing text files and Ada source programs. The environment also supports interactive syntactic and semantic assistance for developing Ada programs, interactive unit and system testing through a source level Rational debugger, and configuration management and version control. As the Rational environment currently exists at AFIT, however, graphical front end support for the software system life cycle is not available. A Rational Design Facility (RDF) would provide the necessary requirements analysis and design capabilities which would make the Rational

environment an APSE, but this RDF is not currently available at AFIT (59). The environment does provide for process management so that project managers can determine a project specific environment.

The Network. A network is utilized which allows the aforementioned environments to communicate. The network employs sixteen servers including one Everex 386 33 MHZ machine and fifteen Zenith 386 25 MHZ machines. The Computer Engineering Department utilizes the Everex and a Zenith as file servers. The network provides for a distributed environment. This distributed environment allows a software developer to employ the capabilities of numerous environments for the development of one software system. For example, a user may use Excelerator to analyze and design a software system in the PC environment, then use another environment such as UNIX or VMS to generate the code for their design. Files can be transferred between the various environments using the file transfer tools available. Conversely, the software developer may choose to perform all the software development functions in one environment, for example, VMS.

3.2 CASE Tool Classification

The CASE tools under consideration were classified based on the types and categories for CASE tools provided in chapter two of this document.

DECdesign, POSE and ObjectMaker can be categorized as system design and verification tools since they provide the capability for either object oriented or functional analysis and design of software systems. As such, these tools can be further categorized as front end, upper CASE, or vertical tools.

3.3 Criteria to Evaluate CASE Tools Sets

The criteria used to evaluate the CASE tool sets that were under consideration were largely derived from the Software Engineering Institute's 1987 technical report "A Guide to the Classification and Assessment of Software Engineering Tools" (17:19-30). Other sources were used as well (4:74-80; 20:65-66; 22:191-193; 34:343-357; 47:1104; 52:14-15; 53:6; 62:127-128). Criteria presented in these documents applicable to the AFIT SEE software development process and methods were used.

The criteria used were grouped into several categories of top level criteria. Each category contained lower level criteria that fell within the top level category. A description of these criteria is provided in Appendix B. Each top level criterion was assigned a numeric weight representing its level of importance as determined by AFIT software engineering instructors. A scale of 0-10 was used where the value 0 indicated minimal importance and the value 10 indicated maximum importance. Weighting the criteria allowed

for use of a methodology to evaluate CASE tools which will be discussed later in the chapter. The resulting weighted criteria chart is provided in Appendix C.

Two questionnaires were then developed for use by CASE tool set evaluators, one for the novice CASE tool evaluator, and one for the expert who has had experience with the tool. Evaluators were asked to rate the support given to each top level criterion by the CASE tool they were evaluating. A scale of 0 to 10 was used with a 0 indicating no support for the top level criterion, and a 10 indicating maximum support. An example of the evaluations developed for each group is provided in Appendix D. The collective results of the evaluations were then used to evaluate and compare the CASE tools sets.

3.4 Evaluation Approach

The approach taken to evaluate the tool sets began with an examination of the process for evaluating CASE tools presented in chapter two. This involved examining the first three steps of the process, the needs analysis, the environment analysis, and the development of a CASE tool candidate list.

Step one in the evaluation approach involved a needs analysis to determine the purpose for which a tool would be needed. This required addressing the questions posed in chapter two concerning needs analysis.

In answer to question one, "What model of software development is used by the organization?", no model really exists since AFIT is an academic environment as opposed to a software development organization. Engineering graduate students apply software development methodologies, presented as part of the curriculum, as required. These methodologies can be categorized as functionally-oriented methodologies or object-oriented methodologies.

Question two asks "What major tasks are required by the model?" Again, no set model of software development exists. Graduate students are tasked to apply the methodologies presented to software engineering class projects.

Question three, "Which tasks could be better performed with automated tools?" and question four, "Which tasks are lacking in adequate tool support?" lend themselves more to a software development organization where large software systems are produced. Software engineering products produced by AFIT graduate students are scoped so that quarterly class requirements can be satisfied. These students are reluctant to spend the time required to learn how to use an automated tool to perform a given software engineering task.

This leads to question five, "What are the perceived benefits to be obtained from specific new tools?" Since students are reluctant to take the time to learn how to use a CASE tool, it should be stressed that learning the tool for the purpose of using it on a class project is not as important

as being introduced to the tools of the software engineering professional and becoming educated about the benefits and potential pitfalls concerning CASE tools (as discussed in chapter two). In addition, graduate students may find the tools to be beneficial when used as part of larger projects, i.e. thesis work.

The second step of the evaluation approach involved an analysis of the organizational environment. As mentioned earlier, AFIT is an academic environment where no one particular model of software development, or software development process exists. Instead, students are taught various software development methods with emphasis placed on functionally oriented and object oriented methods. Students and staff require an automated tool set that will assist in performing the tasks required during each activity of the software development process.

The third step of the process was simplified since the candidate CASE tool list was already established prior to the thesis effort. Experienced software engineering instructors had chosen tools which they determined to be potential candidates for use at AFIT. However, now that a method for tool evaluation has been established, it can be used on future potential candidate tools as well.

The fourth stage of the CASE tool evaluation process, applying criteria and selecting a CASE tool set, required the most effort. As already mentioned, criteria had to be

established that pertained to the AFIT SEE. These criteria were then incorporated into questionnaires. To apply the criteria for the purpose of evaluating the CASE tools required the implementation of comprehensive test cases. These test cases fully tested the established criteria and were representative of applications that are typical for the AFIT environment. The establishment of these test cases involved consultation with AFIT software engineering and information systems management instructors who provided test cases which were actual assignments given to their graduate students. These same graduate students, as well as students enrolled in the Professional Continuing Education (PCE) software engineering short courses, voluntarily participated in the evaluation effort. Since the CASE tools under consideration were upper CASE, the test cases used by the students were constructed for software analysis and design using functional and object oriented methods. The test cases are presented in Appendix E.

3.5 Methodology for Software Evaluation and Selection

Upon completion of the CASE tool set forms, a methodology for software evaluation and selection was required to quantify the tool sets' performance for evaluation and selection. Three software selection methodologies were considered (2:707; 24:508; 34:88-101). The methodology chosen, previously used for software other than software applications, was modified for the specific purpose of selecting software packages

(2:707). This methodology was chosen because it provides a measurable means for evaluating CASE tools through the creation of a ratings matrix that can be maintained and modified and used to determine how a CASE tool rates in comparison to other similar CASE tools already incorporated in the AFIT SEE. An evaluator would be able to determine in a glance how a CASE tool under consideration compared with tools already in use at AFIT.

With this methodology, three measures of software quality were considered:

> The frequency with which the attribute ratings of one package exceeded those of another, the presence of outliers, where very poor performance may exist on a single attribute and be glossed over by compensatory methods, and the cumulative magnitude of attribute ratings on one package that exceed those on others. (2:707)

Implementing the methodology involved a six step process. A general description of the six step process (2: 708-710) is as follows:

1) Determine the frequency of occurrences in which comparisons of a package's important attributes are rated greater or equal to another package's same important attributes.

2) Identify a single outlier, if any, where one package is inferior to another. This prevents very good performance on most attributes from disguising very poor performance of one attribute that could substantially impair utilization.

3) Determine the collective, overall magnitude to which the ratings of one package are superior to another. The

larger and more frequent the positive differences between attribute ratings of the packages, the more likely one package is superior to another.

4) Compute the pairwise ratings of the packages.

5) Compute Kendall scores for all the packages and use these scores to rank order the packages from maximum to minimum, where maximum is the highest quality.

6) In the case of a tie, recompute the figures for the packages that tied.

The results and analysis of the evaluation and selection processes are examined in chapter five.

3.6 Summary

The SEE as it exists at AFIT can best be described as both a general and distributed environment. The combined environments that compose the SEE provide support for all stages of the software systems life cycle. Since AFIT is an academic environment rather than a software development organization, no software process model is established and no one method is employed. Instead, students are taught various methods of software development. Thus, the criteria developed to evaluate the tools under consideration, the CASE tool evaluations, and the test cases used were established based on the SEE and the various methods presented. To analyze the results of the evaluations required the use of a methodology. The efforts involved in evaluating the CASE tools and the

difficulties encountered in the process are presented in the following chapter.

IV The Evaluation Process

This chapter examines the process of evaluating the CASE tools under consideration. It begins with an explanation of how the evaluation forms were distributed. Next, the test cases used to test the functionality of the tools are examined. Finally, problems encountered during the evaluation process are discussed. The selection process is discussed in the next chapter.

4.1 Distribution of Evaluation Forms

Two types of evaluation forms were distributed to the evaluators who assisted in this research effort: a form designed for experts and a form designed for novices. Evaluation forms had to be developed which reflected the CASE tool user population at AFIT; experts, such as staff or experienced students, who had previous and substantial CASE tool experience, and novices, such as students who had never used a CASE tool. The results of these evaluations were analyzed and the results of the analysis were used to make recommendations concerning the evaluated CASE tools (discussed in the next chapter). An example of these evaluation forms can be found in Appendix D.

Expert Evaluation Forms. These forms were designed for evaluators experienced in the acquisition and use of CASE tools, and who were familiar with the CASE tools under consideration as well as the environment in which they might

be used. Detailed questions concerning tool support for the criteria presented were asked which required knowledge of the software development process, the various methodologies employed, and the role of CASE tools in a SEE.

Copies of these evaluation forms were distributed to the software engineering staff for the purpose of evaluating POSE. These instructors were familiar with the tool and its capabilities, having incorporated it into their curriculum for use by their students. Since no DECdesign or ObjectMaker CASE tool expert was resident at AFIT during the time of this research, the researcher assumed the role and completed the evaluations. This involved interviewing the systems staff responsible with acquiring and loading the tools onto their systems, the vendors responsible with providing the tools and support required, and instructors who had some familiarization with the tool. It also involved research of ObjectMaker, DECdesign, and Digital manuals as well as testing of the tools by the researcher.

Novice Evaluation Forms. These forms were designed for evaluators with little or no experience using CASE tools. Questions were designed to acquire the evaluator's opinion of the CASE tool's performance and ease of use. The novice evaluation form was considerably less detailed than the expert evaluation form.

Novice forms were distributed to students who used the tool to support their various class projects. AFIT

engineering students evaluated DECdesign using this form. Logistic and engineering students and the researcher evaluated POSE using this form.

4.2 Test Cases

The test cases used to evaluate the CASE tools were developed by the software engineering instructors at AFIT. These test cases were chosen because they reflected the methodologies presented at AFIT and were used as actual assignments to the students who were evaluating the tools. The problem statements for these test cases are presented in Appendix E.

The first test case involved developing an automated college registration system. Students in the software analysis and design course used this test case to test the functional analysis and design capabilities of POSE and DECdesign.

The logistics students developed their own test cases, approved by their instructor, to test the data modeling capabilities of POSE. Since there were a number of different test cases developed by the students, the problem statements are not included in Appendix E.

The final test was developed by software engineering instructors and was previously used as an assignment. This case involved the development of an automated spell checker. The researcher used this test case to test the functional analysis and design capabilities, data modeling capabilities,

and real time design capabilities of DECdesign, POSE, and ObjectMaker. Since the Yourdon method is presented in the AFIT curriculum, it was chosen as opposed to the Gane & Sarson method as the functional analysis and design method. The researcher also tested the object oriented analysis and design capabilities of ObjectMaker using this final test case.

Due to the short duration of their course, PCE students did not use a specific test case to evaluate POSE. Instead, they were given a demonstration of the tool by their instructors. They then experimented with the tool individually and performed an evaluation based on their use of the tool over a two week period.

Some of the hard copy products generated using the tools are presented in Appendix G.

4.3 Problems Encountered

During the process of evaluating the CASE tools, several problems were encountered. These problems were either tool or tool support related.

Problems Concerning POSE. When the evaluation of POSE began, the tool was implemented using a Zenith 248 PC. As a result of using a 286 based machine, there was insufficient memory available to load several of the modules in the toolkit. This problem was rectified when the Zenith 248s were replaced with UNISYS 386 based PCs.

Another problem encountered with POSE was the lack of protection for a user's files. Although twenty copies of POSE were available at AFIT, fourteen of those copies were loaded on PCs used by the software engineering short course students, and five were loaded on PCs at the Logistics school. Each of these nineteen copies was accessible only on the machine on which it was resident. Any files created and stored on a given PC could only be accessed on that same machine.

The remaining copy was loaded on the AFIT PC network and was available for use by anyone with login privileges. Unfortunately, POSE stores all user created files to a common area that can be accessed by anyone using the PC network. No mechanism is provided to protect these files. This presented a situation where an individual could easily access or corrupt another user's files, intentionally or unintentionally. To deal with this problem, PCE instructors permitted software engineering graduate students access to their classroom PCs when they were not being used by PCE students. Copies of POSE were loaded on each PC as opposed to a network. Anyone using a particular PC could still access any POSE generated files on that machine, but access to these machines could be more easily controlled.

A final problem concerning POSE was the availability of documentation for use by the evaluator. Two complete sets of user's guides were available to AFIT. Both sets were kept by the short course instructors who used POSE as a part of their

curriculum. One of these copies was on loan to an AFIT logistics instructor for the purposes of making copies for his students. Software engineering graduate students interested in using the tool had no documentation readily available. This fact made using the tool less desirable to AFIT engineering students.

Problems Concerning DECdesign. The main problem concerning DECdesign involved processing speed. The tool was originally loaded on a Digital work station used for research purposes. When the tool was invoked on the work station it took several minutes to load. It was determined that a main frame would be required for file management in order to speed the processing time of the tool. Using a VAX mainframe to perform file management yielded an increase in the performance speed of DECdesign, however, during peak periods, processing still took several minutes. Further increases in the processing speed required system administrator intervention.

A second problem encountered involved the conversion of text and graphic files to Post Script form for generation on a Post Script printer. When the tool was used to generate a selected report, the user was able to preview the report which appeared to be formatted correctly, before it was saved. The user then had to use a system conversion command to convert the report file from its original form to Post Script form. When the converted report was generated, the format was incorrect. All text and graphics were located in the lower

left hand side of each page with some overwriting of text occurring. After research into the problem, the system administrator determined that AFIT did not possess the latest version of DECdesign. This created another problem. Access to previously created DECdesign files was not possible during the time required to load the new version of the tool. This occurred at a period of time in the research effort when the CASE tool evaluations were being performed. This deterred potential evaluators from using DECdesign. As a result, only two novice evaluations were completed. This low response prevented the proper implementation of Anderson's methodology for evaluating and selecting CASE tools. This is further discussed in the next chapter.

A final problem encountered with DECdesign involved periodic and arbitrary unintentional exiting from the tool. Several times while creating different views using the tool, the researcher was ejected from the tool. Upon reentry into the tool it was discovered that any changes to the view being edited at the time of ejection were not saved. This problem was attributed to the fact that an older version of the tool was being used.

Problems Concerning ObjectMaker. As previously mentioned, the copy of ObjectMaker available to the researcher was a demonstration copy. A password had to be obtained from the vendor which provided the capability to test the tool's functions.

The documentation provided was limited and included some release notes, a tutorial, and an incomplete user's manual. Instructions for creating diagrams using the various functional or object oriented methods were not included. The researcher had to determine how to create various diagrams through trial and error.

Since he had compatibility problems with the window package on the 386 based PCs and ObjectMaker, the systems administrator originally loaded the tool set on a Zenith 248 PC. The memory space on this machine was limited which resulted in a slow execution time when operating the tool.

Finally, the tool set was not available until the latter stages of the research effort. The work load of the PC systems administrator prevented the loading of the tool on a PC until this time. Additionally, when the tool was first loaded, the system administrator could not unlock the tool to allow for execution in the evaluator mode. Instead, the tool was locked in the tutorial mode. All efforts to unlock the tool were unsuccessful. Rather than continue to try to unlock the current version of the tool, the researcher decided to wait for a version of the tool that had just been released. When this version of the tool was received and loaded, difficulties with the tool continued. The vendor eventually sent a 286 based PC with the tool already loaded for the researcher to use. Due to the time constraints, a cursory examination of the tool was performed.

General Problems. Participation by software engineering graduate students in evaluating the CASE tools was minimal. Only two students attempted to use DECdesign. The low participation was attributed to the aforementioned problems as well as what the students perceived to be an unacceptable learning curve based on the time available to work on their project as well as the size of the projects assigned. However, participation by the logistics students and PCE students was adequate for the purposes of evaluating POSE.

4.4 Summary

Several test cases were implemented to evaluate the performance of POSE, ObjectMaker and DECdesign. These test cases were classroom assignments developed by AFIT instructors based on the software development methodologies presented in their courses. Using POSE and DECdesign to complete these assignments, novice evaluators were able to become familiar with the tools and determine how well a tool supported the presented criteria, which were mostly concerned with tool performance. Additional evaluations were performed by instructors and the researcher. These evaluations were more detailed than the novice evaluations and required a familiarity with software environments, software development processes, and the role of CASE tools based on the environment and process. The results of these evaluations were used in an algorithmic process presented by Anderson to determine how the

tools compared to each other and to establish a basis for recommendations concerning the tools. The results of the evaluations as well as the application of the algorithmic selection process are presented in chapter five.

V Results and Discussion

This chapter addresses research question three, "How do the CASE tools under consideration meet the established criteria for each?", research question four, "How do the tools under consideration compare, if at all, to each other?", and research question five, "Do these tools belong in the software engineering environment at AFIT, and if so, how would these tools be incorporated into the defined environment?" The results of the novice and expert CASE tool evaluations are examined. The average numeric weights assigned to the tool evaluation criteria by AFIT software engineering instructors and the average criteria support rating assigned to the tools by the tool evaluators are presented in the form of matrices. These matrices are utilized in Anderson's six step methodology for evaluating and selecting software. The results of this methodology are examined and a comparison of the CASE tools under consideration is made.

5.1 Evaluation Results

As previously discussed, several software engineering instructors were asked to assign numeric weights to criteria used for evaluating CASE tools under consideration for use in the AFIT SEE. The survey results of the five instructors who responded can be found in Appendix B. Ten evaluators responded to the novice evaluation for POSE, two responded to the novice evaluation for DECdesign, one responded to the

novice evaluation for ObjectMaker, and one evaluator responded to the expert evaluation for DECdesign and POSE. Since the sampling of respondents to the DECdesign and ObjectMaker novice evaluations and the expert evaluation for both DECdesign and POSE was very low, a comprehensive comparison of the tools using Anderson's methodology could not be accomplished. Rather, the evaluation results were used as a means of examining the methodology to determine its worth in evaluating and selecting CASE tools for use at AFIT. The results also provided preliminary insights into the tools evaluated. In order to better convey the capabilities of the methodology, two functional analysis and design tools already existing as part of the tool capabilities layer of the AFIT SEE, Excelerator and NASTEC 2000, were evaluated by a resident expert on the tools.

The average criteria weights and the average rating assigned to each criterion were calculated and then used to construct matrices to be used with Anderson's methodology. These matrices are presented in Figure 5.1, the results of the novice evaluations, and Figure 5.2, the results of the expert evaluations. Figure 5.1 contains a 3 x 17 matrix to be used in Anderson's methodology (the methodology and how the contents of the matrix are used is discussed in the next section). P, representing POSE, D, representing Decdesign, and O representing ObjectMaker are rows one, two, and three of the matrix. C is the criterion number and W is the weight of

importance assigned to each criterion. The numbering scheme for the criteria referenced in Figure 5.1 is presented in Table 5.1.

Figure 5.2 is the 4 x 33 matrix containing the results of the expert evaluations. P, D, C, and W are the same as in Figure 5.1. E (row 3) and N (row 4) represent the criteria support rating assigned to Excelerator and NASTEC 2000. The numbering scheme for the criteria referenced in Figure 5.2 is presented in Table 5.2.

Results of novice evaluations

C	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
W	5.0	8.0	9.0	7.0	8.0	9.0	9.0	5.0	9.0	7.0	7.0	4.0	8.0	8.0	9.0	9.0	9.0
P	3.0	6.0	5.0	3.0	2.0	3.0	5.0	6.0	5.0	7.0	5.0	2.0	7.0	3.0	5.0	6.0	6.0
D	3.0	9.0	9.0	3.0	6.0	9.0	7.0	8.0	7.0	7.0	8.0	8.0	10.0	7.0	9.0	9.0	5.0
O	6.0	6.0	7.0	7.0	5.0	7.0	7.0	8.0	6.0	7.0	8.0	9.0	3.0	8.0	8.0	9.0	7.0

C: criterion number
W: average weight assigned to criterion
P: average criterion support rating assigned to POSE (row 1)
D: average criterion support rating assigned to DECdesign (row 2)
O: average criterion support rating assigend to ObjectMaker (row 3)
Matrix A_{ij}, l = 3, n = 17, i = 1..l, j = 1..n

Figure 5.1 Matrix A for Analyzing the Performance Ratings of CASE Tools (Novice)

5.2 Anderson's Methodology

To use Anderson's methodology of evaluating and selecting CASE tools, the following definitions need to first be

C	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
W	5.0	8.4	9.0	[illegible]	[illegible]	8.2	8.2	[illegible]	[illegible]	[illegible]	8.8	[illegible]	6.0	[illegible]	7.4	5.0	4.6
P	5.0	6.0	8.0	8.0	[illegible]	8.0	[illegible]	[illegible]	[illegible]	[illegible]	5.0	[illegible]	5.0	6.0	8.0	3.0	[illegible]
D	2.0	8.0	7.0	4.0	9.0	6.0	9.0	9.0	8.0	6.0	8.0	9.0	5.0	8.0	6.0	8.0	3.0
E	4.0	5.0	7.0	4.0	7.0	6.0	6.0	6.0	1.0	1.0	7.0	4.0	3.0	6.0	6.0	4.0	4.0
N	1.0	2.0	1.0	1.0	3.0	1.0	2.0	1.0	0.0	0.0	1.0	0.0	2.0	2.0	1.0	1.0	1.0

C	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33
W	9.4	10	8.0	8.6	9.0	9.0	6.2	6.0	6.4	7.6	8.2	5.0	3.8	5.8	6.2	9.2
P	7.0	7.0	6.0	6.0	8.0	8.0	4.0	0.0	5.0	10	7.0	7.0	5.0	7.0	8.0	8.0
D	9.0	6.0	4.0	9.0	9.0	7.0	8.0	8.0	5.0	9.5	10	10	9.0	10	9.0	3.0
E	5.0	4.0	7.0	7.0	7.0	5.0	4.0	7.0	8.0	7.0	6.0	7.0	6.0	7.0	6.0	7.0
N	1.0	1.0	1.0	1.0	2.0	2.0	1.0	0.0	0.0	1.0	4.0	5.0	6.0	2.0	3.0	3.0

Figure 5.2 Matrix A for Analyzing the Performance Ratings of CASE Tools (Expert)

considered (2:708): a_{ij} represents the ratings of the ith CASE tool set in regard to support for the jth criterion, $i = 1, 2, \ldots, l$, and $j = 1, 2, \ldots, n$; $\mathbf{A}_{(l \times n)} = \{a_{ij}\}$ (Figure 5.1 and Figure 5.2). In the case where A is used for analyzing the performance rating of CASE tools as determined by novice evaluators, $l = 3$ since three CASE tools are evaluated and $n = 17$ since 17 criteria are rated so that $A_{(l \times n)} = A_{(3 \times 17)}$ (Figure 1). For the expert evaluations $A_{(l \times n)} = A_{(4 \times 33)}$ (Figure 2). W_j represents the weight or importance assigned to the jth criterion by the software engineering instructors. Since 33 criteria are rated in the expert evaluations, $j = 1, 2, \ldots, 33$ indicating 33 different weights, W_1 through W_{33}, are assigned to the criteria; $s_{ij} = \{k \mid a_{ik} \geq a_{jk},\ k = 1, 2, \ldots, n\}$, i.e. the set of criteria for which the rating received by the ith CASE

Table 5.1 Criteria Numbering Scheme for Novice Evaluations

Criteria	Criteria Number
Tailoring	1
Intelligence/Helpfulness	2
Graphics	3
Predictability	4
Error Handling	5
System/Human Interface	6
Tool Understanding	7
Tool Leverage	8
Tool State	9
Data Dictionary	10
Performance	11
Consistency	12
Self Instrumented	13
Correctness	14
Ease of Use	15
Learnability	16
Documentation	17

tool set is equal to or greater than the rating of the jth CASE tool set, $i = 1,\ldots, 1$ and $j = 1,\ldots, 1$. Using the results of the expert evaluations, sixteen sets were calculated and are presented in Table 5.3.

$|s_{ij}|$ denotes the cardinality (size) of the set s_{ij}. In the case where $i = j$, when the ratings of a tool set are compared to the same exact ratings, the magnitude of s_{ij} will be 33 since each rating is always greater than or equal to itself. The magnitude of the sixteen sets presented in Table 5.3 were calculated and are presented in Table 5.4.

$d_{ij} = \min_{k} \{a_{ik} - a_{jk} | a_{ik} \leq a_{jk},\ k = 1, 2,\ldots,n\}$, the minimum difference of the differences calculated between ratings for the same criterion. Using the expert evaluation

Table 5.2 Criteria Numbering Scheme for Expert Evaluations

Criteria	Criteria Number
Tailoring	1
Intelligence/Helpfulness	2
Graphics - General Capabilities	3
Graphics - Specific Capabilities	4
Predictability	5
Error Handling	6
System/Human Interface	7
Tool Understanding	8
Tool Leverage	9
Tool State	10
Data Dictionary	11
Integration	12
Performance	13
Consistency	14
Evolution	15
Fault Tolerance	16
Self Instrumented	17
Methodology Support - General	18
Methodology Support - Specific	19
Life Cycle Support	20
Correctness	21
Ease of Use	22
Learnability	23
Software Engineering Environment	24
Database	25
Tool History	26
Vendor History	27
Purchase Agreement	28
Maintenance Agreement	29
User's Group/Feedback	30
Installation	31
Training	32
Documentation	33

results, these differences were calculated and are presented in Table 5.5. These differences will be used later in the methodology to identify single outliers that may skew the overall rating assigned to a CASE tool. When $i = j$, the resulting d_{ij} will be zero since each rating subtracted from itself will yield zero.

$Z = (\max_i \max_j a_{ij} - \min_i \min_j a_{ij})$, $i = 1,\ldots,l$, and $j = 1,\ldots,n$, i.e. the difference between the maximum and minimum ratings contained in ***A***. In the case where ***A*** reflects the results of the expert evaluations, $Z = (10 - 0) = 10$, since 10 and 0 are the maximum and minimum ratings assigned. These notations, as well as the values calculated using the results of the expert evaluations, will be used in the following application of Anderson's methodology.

The Frequency of Important Attributes. The first step of the methodology involves determining the number of times that CASE tool set i has a rating equal to or greater than those of CASE tool set j ($a_{ik} \geq a_{jk}$) on criteria with large weights (indicating importance). If this occurs frequently, it is an indicator that CASE tool set i is better than CASE tool set j.

In examining the average ratings assigned to the tool sets by the novice evaluators, it was quickly determined that the ratings assigned to DECdesign were all larger than the ratings assigned to POSE. With this type of results, applying the methodology would not be required because the choice of tools would be obvious. The results of the expert evaluations were used for the purpose of examining Anderson's methodology. It should be mentioned again that the disparity in the number of survey responses between the POSE and DECdesign novice evaluations, as well as the low response rate to the expert evaluations, made it infeasible to use the results of the

Table 5.3 Sets of Superior or Equivalent Criteria Ratings

s_{11} = {1,2,3,...,33}

s_{12} = {1,3,4,5,6,13,15,19,20,23,26,27,33}

s_{13} = {1,2,3,4,5,6,8,9,10,13,14,15,18,19,22,23,24,27, 28,29,31,32,33}

s_{14} = {1,2,3,...,33}

s_{21} = {2,5,7,8,9,10,11,12,13,14,16,17,18,21,22,24,25,26, 28,29,30,31,32}

s_{22} = {1,2,3,...,33}

s_{23} = {2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,18,19,21,22, 23,24,25,27,28,29,30,31,32}

s_{24} = {1,2,3,...,33}

s_{31} = {7,8,11,12,14,16,17,20,21,24,25,26,29,30,31}

s_{32} = {1,3,4,6,15,17,26,33}

s_{33} = {1,2,3,...,33}

s_{34} = {1,2,3,...,33}

s_{41} = {17,25}

s_{42} = {33}

s_{43} = {6}

s_{44} = {1,2,3,...,33}

Table 5.4 Magnitude of Superior Criteria Rating Sets

$\|s_{11}\| = 33$	$\|s_{21}\| = 23$	$\|s_{31}\| = 15$	$\|s_{41}\| = 2$
$\|s_{12}\| = 13$	$\|s_{22}\| = 33$	$\|s_{32}\| = 8$	$\|s_{42}\| = 1$
$\|s_{13}\| = 23$	$\|s_{23}\| = 28$	$\|s_{33}\| = 33$	$\|s_{43}\| = 1$
$\|s_{14}\| = 33$	$\|s_{24}\| = 33$	$\|s_{34}\| = 33$	$\|s_{44}\| = 33$

Table 5.5 Maximum Differences Between Compared Criteria

$d_{11} = 0$	$d_{21} = -5$	$d_{31} = -4$	$d_{41} = -9$
$d_{12} = -8$	$d_{22} = 0$	$d_{32} = -7$	$d_{42} = -9$
$d_{13} = -7$	$d_{23} = -4$	$d_{33} = 0$	$d_{43} = -8$
$d_{14} = 0$	$d_{24} = 0$	$d_{34} = 1$	$d_{44} = 0$

evaluations to make strong recommendations concerning the tools. Instead, the results were used to demonstrate the application of Anderson's methodology and to provide preliminary insights into the tools evaluated.

To reflect the frequency of important attributes, a matrix $F_{(1x1)} = \{F_{ij}\}$ is constructed where

$$F_{ij} = \sum_{k \in s_{ij}} W_k \; / \; \sum_{k=1}^{n} W_k, \; i = 1, \ldots, 1 \text{ and } j = 1, \ldots, 1$$

F_{ij} is the value obtained when calculating the sum of the weights assigned to the criteria in set s_{ij} divided by the sum of all the weights. Using the results of the expert evaluations, F_{ij} and $F_{(1x1)}$ were computed using the weights assigned to the CASE tool criteria. $F_{(1x1)} = F_{(4x4)}$ and is shown in Figure 5.3.

To understand the significance of the contents of F, a comparison is made for each tool against all other tools considered in the evaluation. When examining the contents of F, the values in the matrix should be read by comparing the tool in the row being considered against the tool in the column being considered (i.e. $F_{2,3}$ is a comparison of DECdesign

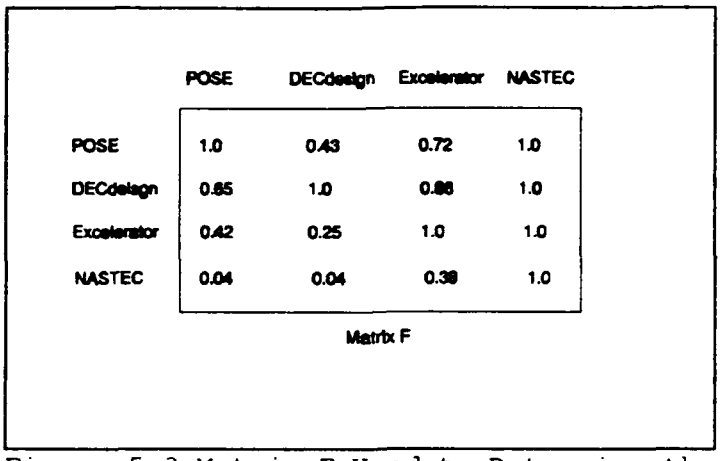


Figure 5.3 Matrix F Used to Determine the Frequency of Important Ratings

to Excelerator). For example, $F_{1,1}$ has a value of 1.0 indicating that when the criteria support ratings received by POSE are compared to the same ratings, the ratings are greater than or equal to the exact set of ratings 100% of the time (as would be expected). The values the evaluator would be concerned with are those that result as a comparison between the ratings of two different tools. For instance, the value of $F_{1,2}$ is 0.43 indicating that 43% of the heavily weighted ratings received by POSE were greater than or equal to the same set of ratings received by DECdesign. $F_{2,1}$ has a value of 0.65 which indicates that 65% of the heavily weighted ratings received by DECdesign were greater than or equal to the same ratings assigned to POSE. An examination of the matrix would suggest a best to last ordering of the tools as follows: DECdesign, POSE, Excelerator, and NASTEC.

In this step, the results suggest that DECdesign would be a better choice for AFIT than POSE based upon the frequency of superior ratings attributed to that tool set as opposed to POSE, and also on the importance (weight) of each superior rating. When the results are not so obvious, however, further measurements must be made to identify the superior tool, if it exists. In addition, outliers may exist which skew the results of this matrix. For these reasons, further analysis of the data is required.

Identifying the Single Outlier. This step identifies a single outlier if it exists, where the ith tool set is inferior to the jth tool set. To show this, $D_{(m \times m)} = \{D_{ij}\}$ is computed where

$$D_{ij} = |d_{ij}|/Z, \; i = 1,\ldots,l \text{ and } k = 1,\ldots,l$$

In this calculation, $|d_{ij}|$ represents the absolute value, not the magnitude. Using the results of previous calculations, the sets $\{D_{ij}\}$ were calculated producing the matrix $D_{(4 \times 4)}$ which is presented in Figure 5.4. When considering each value in the matrix, a comparison should be made of the row tool set against the column tool set (i.e. $D_{2,4}$ should be interpreted as a comparison of DECdesign to NASTEC). For example, the results indicate that when POSE is compared to DECdesign, it indicates that the outlier of very poor performance of greatest significance has an eight point difference when compared to the criterion rating received by DECdesign. When DECdesign is compared to POSE, the outlier of significance has

a five point difference when compared to the criterion rating assigned to POSE.

The matrix values represent the most significant difference between the ratings received by one tool set when compared to the ratings received by another tool set. In the cases where $i = j$ (i.e. $D_{1,1}$, $D_{2,2}$, $D_{3,3}$, and $D_{4,4}$) the matrix value will be zero since the corresponding d_{ij} are all zero. An examination of the matrix would suggest a first to last ordering of tools with the most extreme outliers as follows: NASTEC, POSE, Excelerator, and DECdesign.

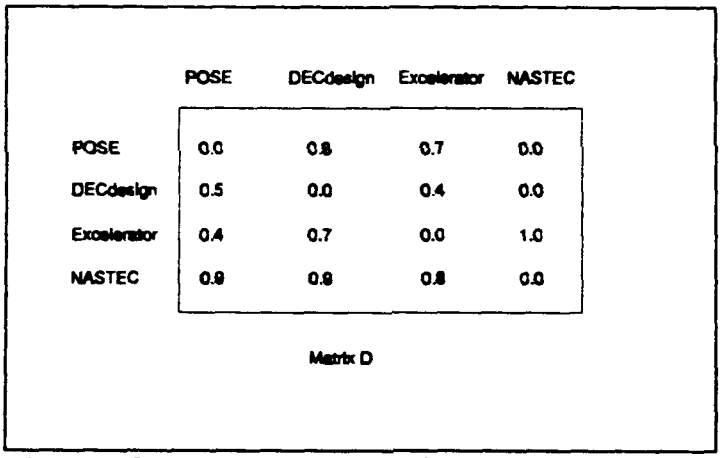


Figure 5.4 Matrix D Used to Determine Outliers

This step is important in that it prevents very low performance of a CASE tool set on one criterion from being overlooked as a result of very good performance on most of the other criteria.

Overall Magnitude of Superiority. In this step, the collective overall magnitude to which the ratings of tool set i are superior to tool set j is calculated. The larger and more frequent the positive differences between the average ratings assigned to the ith and jth tool sets, the more likely it is that tool i is superior to tool j. To measure this, $M_{(1x1)} = \{M_{ij}\}$ was computed using previous calculations where

$$M_{ij} = [\sum_{k \in s_{ij}} (a_{ik} - a_{jk}) / |s_{ij}|] / Z, \quad i=1,\ldots,l, \quad j=1,\ldots,l \text{ and } k=1,\ldots,n$$

M_{ij} is the sum of the differences of the ratings assigned to tool sets i and j, divided by the magnitude corresponding s_{ij}, the set of superior criteria ratings. This value is then divided by Z, the difference of the maximum and minimum criteria support ratings received by all the CASE tool sets. The results of these computations are presented in Figure 5.5.

	POSE	DECdesign	Excelerator	NASTEC
POSE	0.0	0.15	0.08	0.44
DECdesign	0.29	0.0	0.22	0.57
Excelerator	-0.09	0.78	0.0	0.42
NASTEC	-7.25	-18.9	-13.9	0.0

Matrix M

Figure 5.5 Matrix M Used to Determine Overall Magnitude of Superiority

The results suggest that the cumulative magnitude of criteria ratings of DECdesign (0.29 when compared to POSE) exceeds POSE (0.15 when compared to DECdesign), indicating that in this step, DECdesign is the superior tool. An examination of the matrix would suggest a best to worst ordering of the tools as follows: DECdesign, POSE, Excelerator, and NASTEC (the same ordering suggested when examining Matrix F). Again, this matrix alone does not consider all factors when ranking the tool sets. Further calculations are required to determine this ranking.

Pairwise Ratings. In this step, the contents of matrices F, D, and M are used to derive pairwise ratings of the CASE tool sets being considered. To accomplish this, a matrix $P_{(1x1)} = \{P_{ij}\}$ is defined where

$$P_{ij} = \begin{cases} 1, \text{ if } F_{ij} \geq T_s,\ M_{ij} \geq T_m, \text{ and } D_{ij} \leq T_d \\ 0, \text{ otherwise, } i = 1,\ldots,l,\ j = 1,\ldots,l \end{cases}$$

$P_{ij} = 1$ indicates that tool set i is superior to tool set j. The value of P_{ij} is 1 if the frequency of important attribute value in the corresponding F location is greater than or equal to the threshold value T_s, the overall magnitude of superiority value in the corresponding M location is greater than or equal to the threshold value T_m, and the single outlier value in the corresponding D location is less than or equal to the threshold value T_d. The threshold values T_s, T_m, and T_d are determined by the user. For the purpose of this

research, the values used were the same as those used in Anderson's example, T_s = 0.5, T_m = 0.05, and T_d = 0.5. Using the previously calculated values of *S*, *D*, and *M*, Matrix ***P*** is constructed and presented in Figure 5.6. The 1 and 0 values received by each tool are used in the final step of calculating Kendall scores to rank the tool sets. The more 1 values received by a tool set when compared to another tool set, the better its overall ranking. Total values will be calculated by summing the values in a row of the matrix.

	POSE	DECdesign	Excelerator	NASTEC
POSE	0.0	0.0	0.0	0.0
DECdesign	1.0	0.0	1.0	1.0
Excelerator	0.0	0.0	0.0	0.0
NASTEC	0.0	0.0	0.0	0.0

Matrix P

Figure 5.6 Matrix P Used to Rank CASE Tool Sets

Kendall Scores. The contents of matrix ***P*** are then used to rank order the tool sets. This was accomplished by computing the Kendall scores, K_i, for the tool sets where $K_i = \Sigma P_{ij}$. The tool set with the largest Kendall score is considered to be ranked first among tool sets evaluated whereas the tool with the lowest Kendall score is ranked last.

Calculating K_i, $K_1 = 0$, $K_2 = 3$, $K_3 = 0$, and $K_4 = 0$ indicating that DECdesign is ranked first among the rated tools, which all tie for last. In the event of a tie, the final step of Anderson's methodology would be executed. This would involve creating a submatrix of P_{ij} consisting of the rows and columns of the tool set that are tied, then recalculating Kendall scores for that submatrix. If the tie is not broken, the user would randomly assign a rank to one of the tied tool sets and recalculate for any remaining ties. This process would be repeated until no ties are left.

The methodology can now be executed using the results of the novice evaluations. A detailed, step by step explanation of the methodology is not required since this was accomplished using the results of the expert evaluations. The resulting matrices are presented and their contents examined.

The s_{ij} and $|s_{ij}|$ values are calculated and presented in Table 5.6 and Table 5.7 respectively. The d_{ij} values are presented in Table 5.8. Z is calculated to be $Z = 9.5 - 2.7 = 6.8$.

Executing the first step of the methodology produces $F_{(1x1)}$ where $F_{(1x1)} = F_{(3x3)}$. This matrix is presented in Figure 5.7. An examination of the matrix would suggest a best to last ordering of the tools considered as follows: DECdesign, ObjectMaker, and POSE.

Table 5.6 Sets of Superior or Equivalent Ratings (Novice)

S_{11} =	{1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17}
S_{12} =	{ }
S_{13} =	{9,13}
S_{21} =	{1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17}
S_{22} =	{1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17}
S_{23} =	{1,2,3,4,5,6,7,8,9,10,11,13,15}
S_{31} =	{1,2,3,4,5,6,7,8,10,11,12,14,15,16,17}
S_{32} =	{1,12,14,16,17}
S_{33} =	{ 1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17}

Table 5.7 Magnitude of Superior Rating Sets (Novice)

$\|S_{11}\|$ = 17	$\|S_{21}\|$ = 17	$\|S_{31}\|$ = 15
$\|S_{12}\|$ = 0	$\|S_{22}\|$ = 17	$\|S_{32}\|$ = 5
$\|S_{13}\|$ = 2	$\|S_{23}\|$ = 13	$\|S_{33}\|$ = 17

Table 5.8 Maximum Criteria Differences (Novice)

d_{11} = 0	d_{21} = 0	d_{31} = -1.6
d_{12} = -5.9	d_{22} = 0	d_{32} = -6
d_{13} = -6.3	d_{23} = -3	d_{33} = 0

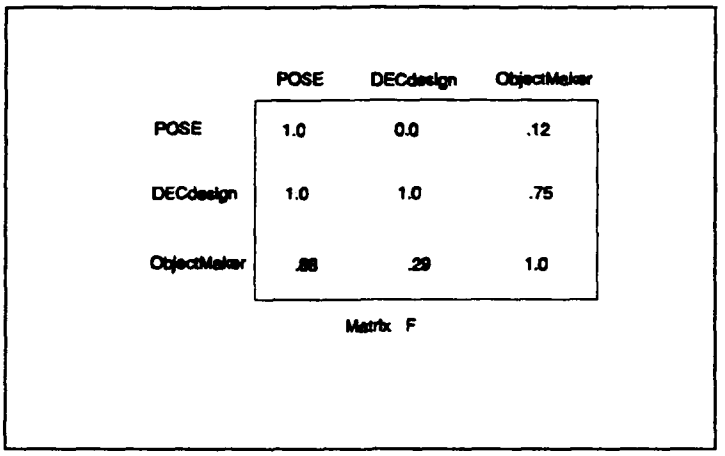


Figure 5.7 Matrix F Used to Determine the Frequency of Important Ratings (Novice)

Executing the second step of the methodology produces $D_{(1x1)}$ where $D_{(1x1)} = D_{(3x3)}$. This matrix is presented in Figure 5.8. An examination of the matrix would suggest a first to last ordering of the tools with the most extreme outliers as follows: ObjectMaker, DECdesign, and POSE.

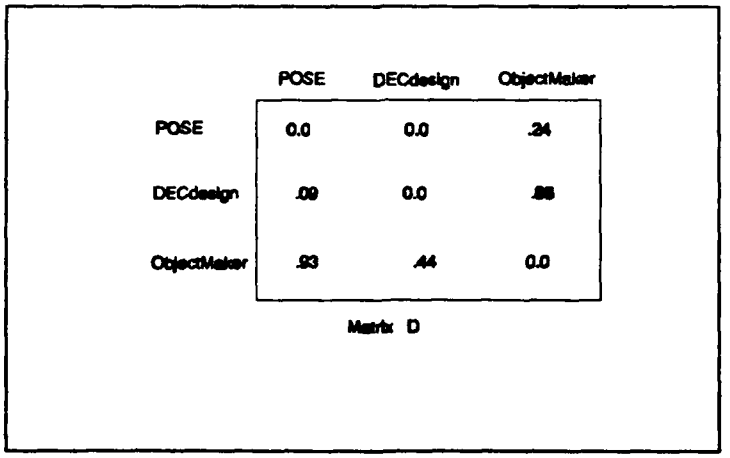


Figure 5.8 Matrix D used to Determine Outliers (Novice)

Executing the third step of the methodology produces $M_{(1x1)}$ where $M_{(1x1)} = M_{(3x3)}$. This matrix is presented in Figure 5.9. An examination of the matrix would suggest a best to last ordering of the tools considered as follows: DECdesign, ObjectMaker, and POSE (the same as the ordering suggested when examining matrix F in Figure 5.7).

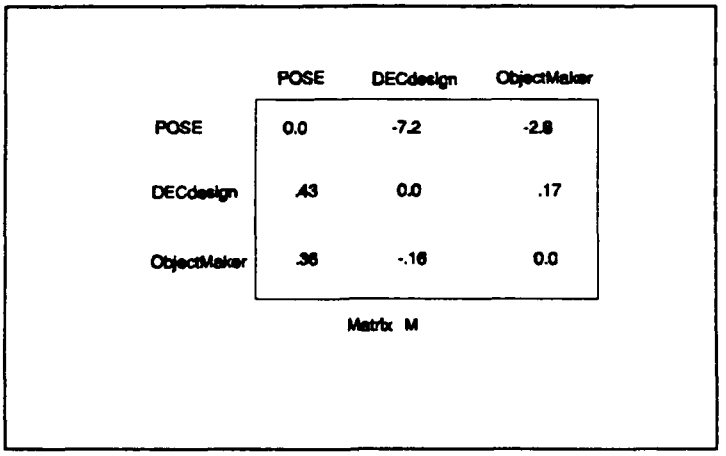


Figure 5.9 Matrix M Used to Determine Overall Magnitude of Superiority (Novice)

Executing the fourth step of the methodology produces the pairwise ratings and $P_{(1x1)}$ where $P_{(1x1)} = P_{(3x3)}$. This matrix is presented in Figure 5.10. When the Kendall scores for each tool are calculated (using the same threshold values as before) DECdesign ranked first among the three tools. A tie breaking step would have to be applied to determine the rankings of ObjectMaker and POSE. Matrices F and M indicate that ObjectMaker may be the superior of the two tools.

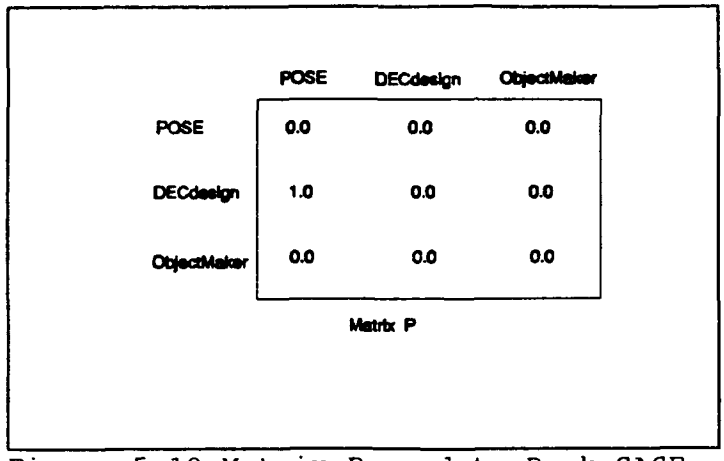


Figure 5.10 Matrix P used to Rank CASE Tool Sets (Novice)

A re-adjustment of the threshold values and a recalculation of the Kendall scores would serve to confirm this.

5.3 Analysis of Individual Criteria

Research question three asks, "How do the CASE tools under consideration meet the established criteria for each?" To answer this question, an examination of the results of both evaluations for each tool considered was conducted. The results of the Excelerator and NASTEC 2000 evaluations were not considered since they were not in the group of tool sets originally considered in the thesis effort.

The results of the novice evaluations indicate that POSE provided a low level of support to a third of the weighted criteria. Error Handling and Consistency were the two criteria that received the lowest ratings, indicating that the novice users thought the tool did not perform well in

recovering from mistakes or in correcting errors. Documentation received a low support rating due to the lack of documentation and manuals available to the users as opposed to poorly written or presented documentation. The tool was given moderate to above moderate ratings for Performance, Ease of Use, and Learnability (which received the highest support rating). These results indicate that novice users felt the tool was easy to use and work with.

The results of the expert evaluation of POSE were more favorable. Only a sixth of the support ratings assigned to the weighted criteria fell below the moderate support level. The expert felt that database support was non-existent and that the tool performed poorly in relation to self instrumentation. The expert agreed with the novice users and assigned high support ratings to Ease of Use and Learnability. The expert felt that the Vendor History was exemplary, reserving the highest support rating for this criterion.

The results of the novice evaluations for DECdesign indicated that the tool provided moderate to almost full support for the criteria considered. These results might have been lower overall with a larger sampling, equitable with the sampling taken for the POSE novice evaluations. They were consistent, however, with the evaluations performed by the researcher on both tools, who found DECdesign to be the more powerful of the two tool sets. Like POSE, DECdesign received the lowest criteria support ratings for error handling and

documentation. At the time of the research effort, only a user's manual was available. Other documentation, such as maintenance manuals or a tutorial, had not been obtained from DEC. This explains the moderate rating the tool received for documentation.

The results of the expert evaluation of DECdesign were not as favorable as the novice evaluation results. Several ratings of weak to below moderate support were assigned to the tool. The tool received a low rating for documentation support for the same reasons attributed to the novice evaluation ratings. The lowest rating was reserved for the tool's support of the tailoring criteria, indicating that the expert perceived that most functions of the tool were consistent and unchangeable. A low rating was given to the tool for support for the life cycle criteria. This was expected as the tool does not support the later phases of the software life cycle. Although the tool received high ratings for its support of the general graphics criterion, it received a less than moderate rating for its support of the specific graphics criterion. This was also expected since the graphics specific capabilities criterion is a comprehensive criterion in that it includes most of the charts, diagrams and tables that are created when utilizing the different software design methodologies presented at AFIT.

An examination of the novice evaluation results for ObjectMaker indicate that the tool provides almost full

support for most of the criteria considered. The lowest criteria support ratings received were for Tailoring, Error Handling, Consistency, and Documentation. The tool provided above moderate support for these criteria. The rating assigned to the Documentation criteria reflects the fact that the documentation was not complete at the time of the evaluation of the tool. The tool did well compared to DECdesign and POSE when considered as a functional analysis and design tool. The ratings received by ObjectMaker would also be applied when comparing the tool to other object oriented analysis and design CASE tools. The results of the novice evaluations indicate that ObjectMaker received ratings that were almost as good as those received by DECdesign, and in a few cases, even better. The results also indicate that ObjectMaker performed much better than POSE as a functionally oriented CASE tool. A comprehensive expert evaluation of the tool would be desirable to confirm the results of the novice evaluations. Based on the results of the novice evaluations, the argument could be made to replace POSE with ObjectMaker as the CASE tool set employed by both PCE and graduate engineering students. Three observations support this argument. First, as already stated, initial evaluation results indicate that ObjectMaker is an overall better choice over POSE as a functionally oriented analysis and design tool. Second, ObjectMaker is not only a functionally oriented tool, it also supports object oriented analysis and design, the two

methods of software development presented at AFIT. Finally, ObjectMaker can be executed on numerous platforms, including PCs and SUN work stations. The purchase of a single copy of ObjectMaker allows AFIT to duplicate as many copies of the tool as required, on any combination of platforms desired.

Although the tool history rated moderate support, the tool received high ratings for its support of criteria that dealt with the vendor, DEC. These ratings were assigned after consultation with the VMS system administrators who had extensive dealings with DEC.

5.4 Comparison of Tool Sets

Although a comprehensive comparison of the CASE tools under consideration was not possible when applying the results of the evaluations to Anderson's methodology, the results were results were more than adequate when used to examine the methodology and provide preliminary insights into the tools evaluated. Using these results, research question four, "How do the tools under consideration compare, if at all, to each other?" was addressed using Anderson's methodology. Further comparisons of the CASE tools were conducted using the results of the expert evaluations without applying the methodology. The fact that there was a low sampling for the expert evaluations was not as important as a low sampling for the novice evaluations. The expert evaluations were designed for experienced CASE tool users, and the questions were straight

forward as opposed to the novice evaluation questions which queried the user for more opinionated answers.

In examining these results, DECdesign received a higher average rating than POSE for twenty three out of the thirty three rated criteria. In ten of these cases the weights assigned to the criteria were high indicating the importance of that criteria in the opinion of the software engineering instructors surveyed. The remaining thirteen criteria were weighted as somewhat important by the instructors. In the ten cases where POSE received a higher rating, seven of the ten ratings were weighted heavily by the software engineering instructors, while three others were weighted as more than somewhat important, indicating that although DECdesign received better criteria support ratings overall when compared to POSE, POSE and DECdesign provided an equitable amount of support for heavily weighted criteria. Considering the expert evaluation data collectively using Anderson's evaluation methodology, DECdesign was considered the better choice of tools in each step and was rated as the top ranked tool.

5.6 Summary

The results of the novice and expert evaluations were utilized to determine the performance of the CASE tool sets under consideration. The results proved useful in demonstrating that Anderson's methodology would be useful in evaluating and selecting CASE tools for incorporation into the AFIT SEE. In addition to examining the results of the

application of the methodology, individual criteria support ratings were reviewed to further determine the worth of the CASE tools to AFIT. Finally, a comparison of the tool sets was made which indicated that, of the tools considered, DECdesign best met the needs of AFIT for a structured analysis and design tool. Again, these results are somewhat suspect due to the low sampling of evaluators. The possibility exists that DECdesign, ObjectMaker, and POSE are all acceptable tool sets, possibly being used effectively on different platforms.

The final research question concerning the place, if any, of the tools in the AFIT SEE, is addressed in the next chapter along with recommendations concerning the SEE, the tool evaluations, and the use of Anderson's methodology.

VI Conclusions and Recommendations

This chapter presents conclusions and recommendations concerning the AFIT SEE, the CASE tools under consideration, and the tool evaluation process are presented as well as recommendations for future studies.

6.1 Concerning the SEE

When considering how the tools belong in the software engineering environment at AFIT, the tools can be grouped as those that already played some role in the AFIT SEE and those that did not. DECdesign and POSE, the functional analysis and design tools evaluated, were already incorporated into the SEE while ObjectMaker, the object oriented tool, was not.

DECdesign is part of an automated tool set provided by DEC to AFIT under contract. What is lacking is documentation to support the tool in the form of user's manuals, maintenance manuals, and interface manuals. To acquire these would require funding from AFIT. Although the documentation would be inexpensive when compared to the cost of the tool itself, there is no need to purchase these manuals. The researcher was able to obtain a tape cartridge of the user's manual from the vendor. From this tape the systems administrator was able to generate a hard copy of the manual which was utilized during the evaluation process. To encourage use of DECdesign by AFIT students and staff, it is recommended that a condensed version of the user's manual be drafted which would be

concerned with those functions of DECdesign that would support the requirements of the AFIT software engineering curriculum. DECdesign supports Yourdon's method for functionally oriented analysis and design of software systems, a method presented at AFIT, and is available to the students and staff. DECdesign also needs to be made available to as many students and staff as possible. Digital has developed a package that will allow DECdesign to execute in the X-windows environment. This package should be obtained and utilized.

POSE has also been incorporated into the AFIT SEE. The tool is utilized in the PCE software engineering short courses and has seen use in the School of Logistics as well. As seen in chapter five, a reasonable sampling of novice evaluations indicated that POSE received moderate to above moderate ratings for heavily weighted AFIT CASE tool support criteria. Like DECdesign, POSE supports Yourdon's method. POSE also has a voluminous user's manual that is time consuming to become familiarized with. As with DECdesign, it is recommended that a condensed user's manual be drafted for use by software engineering graduate students. Also, more copies of the tool set need to be obtained for use by AFIT students. The potential for the expansion of the role of POSE in the AFIT SEE exists. The tools could be used by software engineering graduate students to support functionally oriented analysis and design projects. The initial indications are that both tools will be useful.

DECdesign and POSE are just two of a number of functionally oriented CASE tools that exist in the AFIT SEE (others are Excelerator and NASTEC 2000, two tools which did not perform nearly as well in the evaluations). What is lacking in the SEE are tools that support the second major software development methodology presented at AFIT, object oriented tools. Although it was not possible to conduct a thorough evaluation of ObjectMaker, the tool shows potential to become a part of the AFIT SEE. ObjectMaker is one of the few tools available that supports a number of object oriented analysis and design methods, including those presented at AFIT, such as Coad diagrams and Buhr diagrams. As previously indicated, the tool can be executed on several different types of platforms. Only one copy of the tool must be purchased by AFIT. As many additional copies as required can then be generated. In addition, the tool provides support for the implementation stage of the software systems life cycle. The user can create Buhr diagrams which the tool will transform to outlined Ada packages. The need exists for an object oriented tool in the AFIT SEE. It is recommended that a list of candidate object oriented tools be compiled and that these tools be evaluated using Anderson's methodology to determine which tool would best fit into the SEE. Preliminary results seem to indicate ObjectMaker will be worth trying. This would help AFIT obtain more data upon which to base a decision of whether or not to keep POSE or replace it with ObjectMaker (as

discussed in chapter 5). Of course, other object oriented tools should be evaluated, but perhaps it would be most useful to identify candidates that use other than PC platforms. It is also recommended that a comparison of POSE and ObjectMaker be performed since both tools share the same platform.

Concerning the SEE, a major concern has already been addressed in this chapter, the need for object oriented CASE tools. If staff and students are to use CASE tools to support various software engineering projects, tools must be available that support the major software development methodologies presented at AFIT. The defined SEE should be monitored and periodically evaluated to insure that the environment continues to provide the support required at AFIT. This is necessary due to the evolving nature of any SEE.

6.2 Concerning Anderson's Methodology

The question might be posed "If the CASE tools already exist in the AFIT SEE, why evaluate them?" The answer is the need to establish a baseline ratings matrix with which to rate and compare all candidate tools. Anderson's methodology provides a means of determining the important attributes of a CASE tool being considered for use at AFIT and then rating the candidate tool against existing tools as well as other candidate tools. More importantly, the methodology provides a measurable means of determining if a candidate tool should be incorporated into the AFIT SEE. Conversely, it can also be used to determine if existing tool sets can be replaced by

superior tools. For example, an examination into the worth of the NASTEC 2000 and Excelerator CASE tools as continuing components of the AFIT SEE may be desirable.

The rating matrix created as a result of applying the methodology would be modified each time evaluations were performed on candidate tools. The matrix would then continue to grow as each new tool is evaluated. Separate ratings matrices should be created for CASE tools dependant on the methodology they support, i.e. a ratings matrix for functionally oriented CASE tools and a ratings matrices for object oriented CASE tools. Different matrices should also be created for CASE tools dependant on the various stages of the software systems life cycle supported by a tool, i.e. a ratings matrix for Upper Case tools and a matrix for Lower CASE tools. Two versions of the aforementioned matrices should be created, one reflecting the results of novice evaluations and one reflecting the results of expert evaluations. These matrices would provide a variety of results from which to select a candidate tool.

As the software engineering instruction staff and academic requirements for software engineering students change, the weights assigned to the CASE tool performance criteria should be recalculated. This will keep the ratings matrix current with the reflected views of the current instructing staff. It is recommended that an acceptable sampling of evaluations of existing CASE tools be conducted so

that baseline ratings matrices can be established. This would include Upper and Lower CASE tool ratings matrices and functionally oriented and object oriented CASE tool matrices with novice and expert versions of each. Any student or staff member tasked with evaluating candidate tools could then apply Anderson's methodology using the pre-established matrices to determine the tool's worth.

6.3 Concerning the Evaluations

As they exist, the expert and novice evaluations used to provide the CASE tool criteria support ratings required to implement Anderson's methodology are quite comprehensive. A large number of criteria are examined for the purpose of performing as thorough an evaluation as possible of the tools under consideration. Although this is an extensive list of criteria, it may not be complete. AFIT software engineering instructors should be polled periodically to determine the completeness of the criteria list as well as the worth of each individual criterion. The grouping of the criteria in the evaluations should also be examined to determine if the criteria might be better categorized (i.e. the establishment of a security criterion or a grouping of methodology support by specific methodologies). Both expert and novice evaluations should be maintained to account for the opinions of users with varying levels of CASE tool experience. These evaluations should be used to establish the baseline ratings

matrices, as well as any future modifications to those matrices.

6.4 Summary

This thesis identified the SEE as it exists at the AFIT School of Engineering. Recommendations to improve the SEE were made, including the suggestion that object oriented CASE tools be incorporated into the SEE at the tool support layer. A CASE tool evaluation methodology was presented that provides a measurable means of rating existing and candidate CASE tools for incorporation into the AFIT SEE. It was recommended that a collection of ratings matrices be developed and maintained based on the category of the CASE tools, the software development method the tools support, and the level of expertise of the CASE tool evaluator. Criteria to evaluate CASE tools for use in the AFIT SEE were established and weighted based on individual criterion importance. CASE tools under consideration were rated on how well they supported these criteria by both expert and novice evaluators. It was recommended that the criteria and evaluations be reviewed and updated as required.

Appendix A: Description of Evaluated CASE Tools

The following descriptions of DECdesign, POSE, and ObjectMaker were extracted from the user's manuals provided with the tools.

DECdesign

DECdesign is a CASE tool developed by Digital. The purpose of DECdesign is to provide a Digital-developed graphics environment for analyzing, designing, prototyping, and automatically generating applications. DECdesign supports three of Digital's major application development programs: CASE, database systems, and transaction processing.

DECdesign is a VMS-based DECwindows environment for the graphical analysis and design of applications of any size and complexity. It is the first phase of the DECdesign program.

DECdesign features include:

Integrated process, data, and real-time modelling support based upon Yourdon, Gane & Sarson's, Extended Entity Relationship and Ward-Mellor techniques.

Validation - DECdesign automatically checks the syntactical and semantical design errors of the analysis and design effort.

Version control - Multiple versions of analysis and design models are supported.

Data sharing - Teams of users can have ready access to all the latest designs and design changes.

Load/extract capability - The tool allows designs to be shared across multiple projects and systems.

Single database - All analysis and design information is in one location to simplify maintenance and reuse of data.

CDD/Plus Repository integration - Provides capability to reuse data definitions (stored in CDD/Plus) from different applications and provides updates to or from CDD/Plus repository when changes occur in either DECdesign or CDD/Plus.

Reports - The tool provides extensive reporting of information captured during analysis and design.

Client/server support - Analysis and design models can be shared among networked workstations allowing better CPU utilization.

Online training - The tool provides interactive online product training.

DECdesign also has the capability to integrate with LSE/SCA's Program Design Facility. Information can be shared between DECdesign windows and LSE/PDF windows.

Picture Oriented Software Engineering Tool Set

Picture Oriented Software Engineering (POSE) is a CASE tool developed by Computer Systems Advisors INC., and was designed with three main functions in mind. They are information systems planning, business area analysis, and systems design (using functional design techniques). POSE is composed of ten separate modules. Each module is a stand alone module and can be invoked by using a mouse to select the icon for that module from a main selection screen. A description of each module is as follows:

Data Model Diagrammer (DMD) - POSE-DMD allows the user to graphically represent (by means of entity relationship diagrams) any business system under study in the form of a data model. After data about the business is captured in ERDs and descriptive forms, (data dictionaries), POSE-DMD analyzes the data to determine the existence of foreign keys and redundant attributes, and generates all possible associations implied by the data.

Data Model Normalizer (DMN) - POSE-DMN allows the user to normalize the data attributes of the data model created in DMD. It removes repeating groups, removes partial dependencies, and then removes indirect dependencies.

Logical Database Design (LDD) - POSE-LDD allows the user to define a transaction, its data access and frequency. LDD analyzes the load on various transactions and their access paths so as to optimize the logical database design. It enables the user to make better decisions when transforming a data model into a logical database design.

Data Base Aid (DBA) - POSE-DBA allows the user to select the logical data model that has been captured in POSE-DMD and translate it into a physical design of the target Database Management System (DBMS) under the DBA system.

Data Flow Diagrammer (DFD) - POSE-DFD lets a user design a system in terms of its processes and data flows. It adopts the diagramming convention of the Gane and Sarson methodology.

Structure Chart Diagrammer (SCD) - POSE-SCD allows the user to represent the hierarchy of the program modules and interfaces between the modules and interfaces between the modules in the form of a structure chart. SCD performs consistency checks (with respect to the Yourdon and Constantine Technique) that assists the user in producing a program design with weak coupling and strong cohesion.

Decomposition Diagrammer - (DCD) - POSE-DCD allows the user to create decomposition diagrams, a chart that is used to show functional breakdowns. The three categories of functional decomposition are process decomposition, work breakdown structure, and organizational charts.

Action Chart Diagrammer (ACD) - POSE-ACD enables systems designers to draw action diagrams which can be used to depict program overviews and detailed logic. (Pseudo code and structured english).

Screen Report Prototyper (SRP) - POSE-SRP allows the user to design and develop data entry and menu selection screens as well as reports for a particular application. Planning Matrix Diagrammer (PMD) - POSE-PMD is a matrix tool that allows the user to document and diagram the relationship between two classes of data in the form of a matrix.

The user can use POSE for the Information Systems Planning phase by using PMD and DCD to identify the objectives and strategies of the organization by defining the functions that must be performed and their relationships. Then, DMD and DFD are used to create high level data and process architectures that will support the organizations objectives.

In the Business Area Analysis phase, DMD, DFD, and DMN are used to refine high-level architecture of the system. For data, DMD is used to further analyze information requirements, analyze current data, and decompose global data requirements into business area models. DMN will develop a normalized business area data model. DFD will be used to analyze the flow of information and reduce the complexity by creating multiple-level data flow diagrams. PMD will be used to define and describe the relationships which exist between data and processes, and SRP will be used to construct a prototype of the system.

For Systems Design, the data model created from the business area analysis is used to create the logical and physical designs of the system. SCD is used to design program structures and ACD will turn those structures into complete program specifications ready for implementation.

ObjectMaker

ObjectMaker, an analysis and design CASE tool set developed by Mark V Systems, offers the following functionality:

- Object oriented analysis
- Structured analysis
- Behavioral (state) analysis
- Forms entries for each supported analysis method
- Summary table views for each analysis method
- Object oriented design
- Structured design
- Behavioral (state) design
- Forms entries for each supported design method
- Summary table views for each design method
- A common semantic repository linking all methods
- Ada, C++ and C code generation from detailed design diagrams
- Reverse engineering from Ada, C++ or C source code
- Diagram export support for popular publishing software including Word Perfect and Microsoft Word

The tool provides off-the-shelf support for over twenty analysis and design methods and notations, both object oriented and traditional, including:

ADARTS	Coad-Yourdan	Hatley-Pirbhai
Bailin	Colbert	Martin
Berard	Constantine	Shlaer-Mellor
Booch86	DeMarco	SPS
Booch91	Firesmith	Ward-Mellor
Buhr84	Gane-Sarson	Yourdon
Buhr90	GE	R-Nets
Chen	Harel	F-Nets

ObjectMaker incorporates an externalized rules-based architecture that allows users to easily adapt this tool to individual, project and corporate requirements. The ObjectMaker Tool Development Kit provides extension language programming which may be used to:

- Modify menus and accelerator Keys
- Modify existing method rules and notations
- Create custom methods including rules and semantic mapping
- Modify or create forms entries to the semantic repository
- Develop custom text generation from diagram routines
- Develop custom code generation directly from diagrams
- Map ObjectMaker to an external framework or environment

The Tool Development kit lets the user build a custom tool.

ObjectMaker operates within three graphical user interfaces; Microsoft Windows 3.0, X-Windows R11.4 and the Macintosh. Specifically, ObjectMaker runs on the IBM (or compatible) 286, 386, and 486 platforms and the Apple Macintosh. In the workstation class of systems, ObjectMaker has been ported to DEC (VMS and ULTRIX), APOLLO, Data General, Evans and Sutherland, IBM RS6000, MIPS, and Sun Microsystems. Distributed environments are supported via TCP/IP.

In addition to a mouse pointing device (or compatible alternative), Object Maker typically requires the following memory and hard disk storage:

Operating system	CPU memory	Hard disk space
Microsoft Windows	2 Mbytes	5 Mbytes
Macintosh	4 Mbytes	5 Mbytes
X-Windows	8 Mbytes	10 Mbytes

Appendix B: Criteria Description

The following is a description of the criteria used to evaluate the CASE tools under consideration:

Tailoring: Refers to the extent to which the user interface of the program may be altered to conform to the preferences of the user.

Intelligence/Helpfulness: Refers to a tool's capability to perform functions without the user having to directly specify their initiation, as well as anticipating the user's interaction by providing simple and efficient means for executing functions the user requires.

Graphics - General Capabilities: Refers to a tool's ability to support general interface and output graphics functions required by the user.

Graphics - Specific Capabilities: Refers to a tool's ability to provide graphics functions which support specific software development methodologies as required by the user.

Predictability: Refers to a tool's ability to respond to the user with command names that suggest function.

Error Handling: Refers to a tool's capability to be tolerant of user errors as well as the ability to check for and correct these errors.

System/Human Interface: Refers to a tool's capability to interact with one user, many users, or other tools.

Tool Understanding: The extent to which a tool understands the inner structure of objects the tools manipulates as well as the structures content, and the ability of the tool to handle more detailed or more general aspects of that structure.

Tool Leverage: The extent to which small actions by the user create large effects. The ability of the tool to allow user defined macros or to define a command as an action to be applied to a specific object so that the same command name can have a different implementation for different objects.

Tool State: Refers to a tool's capability to remember how it has been used in a current session or in a previous session.

Data Dictionary: Refers to a tool's ability to create, maintain and update a data dictionary or interact with an existing data dictionary.

Integration: Refers to a tool's ability to be an integrated part of a SEE.

Performance: Refers to a tool's ability to function efficiently and be responsive to the user.

Consistency: Refers to the consistency of a tool's operations and performance.

Evolution: Refers to a tool's ability to evolve over time to accommodate changing requirements, changes to the environment, correcting detected flaws, and performance enhancements.

Fault Tolerance: Refers to a tool's ability to avoid irreparable damage as a result of any environmental failures.

Self-Instrumented: Refers to a tool's ability to assist in determining the cause of a problem once the symptom has been detected.

Methodology Support - General: Refers to how well a tool supports and automates software development methodologies.

Methodology Support - Specific: Refers to how well a tool supports and automates specific software development methodologies supported at the software organization.

Life Cycle Support: Refers to how well a tool supports and automates the different stages of the software systems life cycle.

Correctness: Refers to a tool's ability to operate correctly and produce correct output.

Ease of Use: Refers to a tool's ability to provide user friendly functions and actions.

Learnability: Refers to a tool's ability to provide a command set that is consistent and understandable, as well as the tool's ability to interact with the user to help learn to use the tool properly.

Software Engineering Environment: Refers to how well a tool fits into a SEE.

Data Base: Refers to a tool's ability to create, maintain and update a data base or interact with an existing data base.

Tool History: Refers to a tool's track record.

Vendor History: Refers to the vendor's reputation and track record.

Purchase, Licensing, or Rental Agreement: Refers to the legal agreements associated with a tool.

Maintenance Agreement: Refers to the maintenance support provided for a tool.

User's Group/User Feedback: Refers to a group that has experience with a tool and can share information with a user.

Installation: The installation support and responsibilities of the vendor and a user.

Training: Refers to the existence of and levels of training provided a user by a vendor concerning a tool.

Documentation: Refers to the documentation provided to a user concerning a tool.

Appendix C: Weighted Criteria for Evaluating CASE Tools

The following are the results of a survey distributed to AFIT software engineering instructors concerning criteria for evaluating CASE tools. Instructors were asked to assign a numeric weight to each criterion based on their view of the importance of that criterion in selecting CASE tools to be incorporated into the AFIT SEE. The weight scale is a 0-10 scale with a 0 indicating the criterion was not required, while a 10 indicated that the criterion was absolutely required to evaluate the CASE tools under consideration. The average of the weights assigned were then used to assist in determining recommendations concerning these CASE tools. The criteria and averaged weight values are presented below.

Criteria	Average Weight
Tailoring	5.0
Intelligence/Helpfulness	8.4
Graphics - General Capabilities	9.0
Graphics - Specific Capabilities	9.5
Predictability	7.8
Error Handling	8.2
System/Human Interface	8.2
Tool Understanding	8.2
Tool Leverage	5.2
Tool State	3.7

Criteria	Average Weight
Data Dictionary	8.8
Integration	6.4
Performance	6.0
Consistency	7.6
Evolution	7.4
Fault Tolerance	5.0
Self-Instrumented	4.6
Methodology Support - General	9.4
Methodology Support - Specific	10.0
Life Cycle Support	8.6
Correctness	9.0
Ease of Use	9.0
Learnability	9.0
Software Engineering Environment	6.2
Data Base	6.0
Tool History	6.4
Vendor History	7.6
Purchase, Licensing, or Rental Agreement	8.2
Maintenance Agreement	5.0
User's Group/User Feedback	3.8
Installation	5.8
Training	6.2
Documentation	9.2

Appendix D: Evaluations for CASE Tools

This appendix contains examples of the evaluations used by both expert and novice CASE tool evaluators. The evaluations contain criteria used to evaluate the tools, guidelines to consider when determining the level of support given to the criterion by the CASE tool, and a support rating scale for each criterion.

CASE Tool Evaluation (Expert)

Tailoring

1) Various aspects of the interface can be tailored to suit the user's needs, including application and ability level.

2) The user can turn off unwanted functions that may be obtrusive.

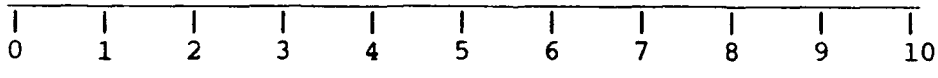
3) The tool's capabilities can be expanded to allow for new functions.

4) The tool's input and output formats can be redefined by the user.

5) Tailoring operations can be controlled to maintain consistency within the using project.

6) The tool can be configured by the user for different resource tradeoffs to optimize such things as response speed, disk storage space, and memory utilization.

Rating Scale



Intelligence/Helpfulness

1) The tool is interactive.

2) The tool prompts for command parameters.

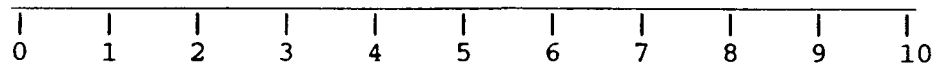
3) The tool prompts for complete command strings.

4) The tool checks for command errors.

5) Action initiation and control is left with the user.

6) Quick and meaningful feedback on the system status and progress of interaction and execution is given the user.

Rating Scale



Graphics - General Capabilities

1) The tool provides the capability to produce hard copy output of diagrams or views created by the user.

2) The objects defined in a diagram are decomposable to as many levels of depth as needed to reach the lowest level process.

3) The tool provides graphics functions such as copy, move, enlarge/shrink, delete/undelete, and zoom in/out for operations on objects.

4) The tool provides the capability to combine or decompose diagrams.

5) The interface is simplified by the use of sound or graphics.

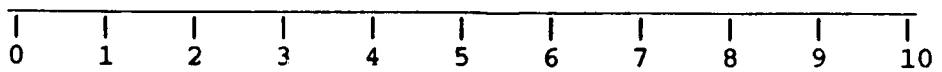
6) The interface includes graphical icons with shape and texture.

7) The interface includes the ability to color code graphical representations.

8) The user can access and retrieve stored information quickly and with little effort while using the system.

9) The tool provides the capability to graphically create user defined symbols.

Rating Scale



Graphics - Specific Capabilities

1) The tool provides the capability to graphically create data flow diagrams.

2) The tool provides the capability to graphically create entity relationship diagrams.

3) The tool provides the capability to graphically create object oriented diagrams (Coad, Booch).

4) The tool provides the capability to graphically create structure charts.

5) The tool provides the capability to graphically create Petri nets.

6) The tool provides the capability to graphically create state transition tables.

7) The tool provides the capability to graphically create control flow diagrams.

8) The tool provides the capability to graphically create flow charts.

9) The tool provides the capability to graphically create Ada package dependency programs.

10) The tool provides the capability to graphically create state transition tables.

11) The tool provides the capability to graphically create object hierarchy or tree diagrams.

12) The tool provides the capability to graphically create block diagrams.

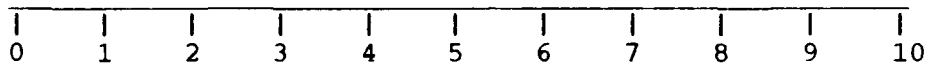
13) The tool provides the capability to graphically create object interaction diagrams.

14) The tool provides the capability to graphically create concept maps.

15) The tool provides the capability to graphically create context diagrams.

16) The tool provides the capability to graphically create structure diagrams.

Rating Scale

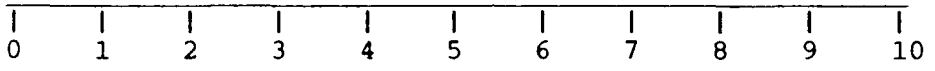


Predictability

1) Responses from the tool are expected in most cases.

2) It is possible to predict the response of the tool to different types of error conditions.

Rating Scale



Error Handling

1) The tool recovers from errors easily.

2) The tool has mechanisms that protect the user from costly errors.

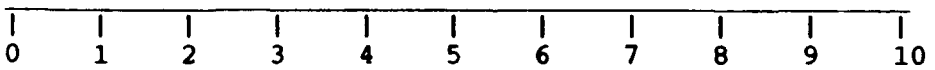
3) The tool executes periodic saves of intermediate objects to ensure that all work is not lost if a failure occurs during a long session with the tool.

4) The tool protects against damage to its database caused by inadvertent execution of the tool.

5) The tool helps the user correct errors.

6) The tool checks for application specific errors.

Rating Scale



System/Human Interface

1) The tool is designed to be used by more than one person at a time.

2) The tool provides for management of work products for single and multiple users.

3) The tool prevents unauthorized access to or modifications of data.

4) The tool provides easy to use menus.

5) The tool allows experienced users to bypass menus.

6) The tool allows use of a mouse or keyboard.

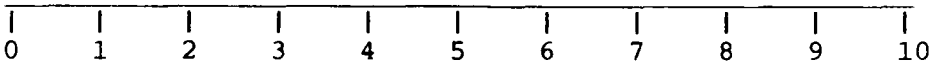
7) The tool provides split screen/window capability.

8) The tool provides undo capabilities.

9) External files can be excepted as input.

10) External files can be produced for outside use.

Rating Scale



Tool Understanding

1) The tool operates on objects at different levels of abstraction or at different levels of detail.

2) The tool can modify collections of objects so as to preserve relationships between them.

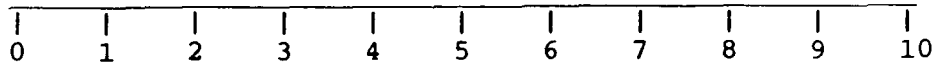
3) The tool can remove objects and repair the structure.

4) The tool can insert objects with proper changes to the structure.

5) The tool will perform validation of objects or structures.

6) The tool will perform consistency checks between levels of detail.

Rating Scale

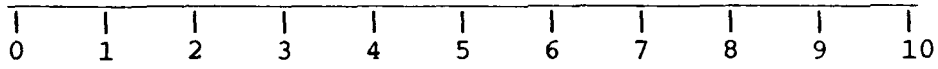


Tool Leverage

1) Commands can be bound to specific object types or structure templates.

2) Commands can be applied systematically to entire collections of similar objects.

Rating Scale



Tool State

1) The tool keeps a command history.

2) Commands can be reinvoked from the history.

3) The command history can be saved to be used with a new run of the tool.

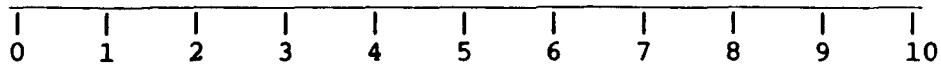
4) Reinvoked command histories can be modified.

5) The current state of the tool and the objects it is manipulating can be saved.

6) The saved state of the tool and the objects it is manipulating can be restored.

7) The tool will keep state across invocations.

Rating Scale



Data Dictionary

1) The Tool dictionary is automatically updated from the design.

2) The tool automatically validates dictionary and design.

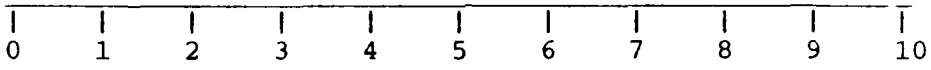
3) Unneeded entries can be deleted from the data dictionary.

4) A rename capability exists that will rename data definitions as well as all references to that structure in diagrams and other structures.

5) Data descriptions can be created to describe in detail in textual form, each data structure and data element.

6) The data dictionary can interface to a host data dictionary.

Rating Scale



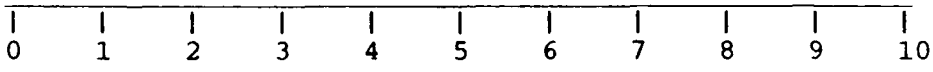
Integration

1) The tool has an effective interface to other CASE tools in the AFIT SEE or under consideration.

2) The tool integrates easily into the AFIT SEE.

3) The interface is compatible with other tools in a set or other commercially available tools.

Rating Scale



Performance

1) The tool will keep and/or employ statistics of command frequency and operand history.

2) The tool's response time to commands will be acceptable relative to the complexity of the operations performed by the command.

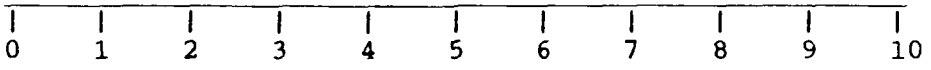
3) A tool supporting multiple users will have a response and command execution time acceptable with the maximum load of users.

4) When the tool is running on the user's hardware it will handle a development task of the size required by the user.

5) The tool provides a mechanism to dispose of useless byproducts it creates.

6) The tool provides a means of prototyping.

Rating Scale



Consistency

1) The tool has well defined syntax and semantics.

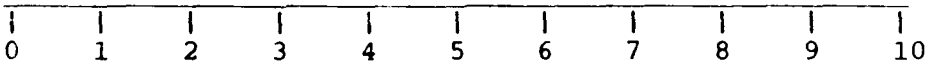
2) The output of the tool can be archived and selectively retrieved and accessed.

3) The tool can operate in a system with a unique identification for each object.

4) The tool can re-derive a deleted unique object.

5) The tool checks for the existence of an object prior to rederivation of the object.

Rating Scale



Evolution

1) The tool is built in such a way that it can evolve and retain compatibility between versions.

2) The tool smoothly accommodates to changes in the environment in which it operates.

3) New versions of the tool will interface with old versions of other related tools.

4) New versions of the tools will operate correctly on old versions of target objects.

5) Old versions of the tool will operate correctly on new versions of the target objects.

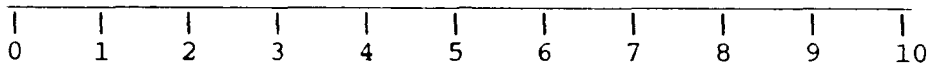
6) Separate versions of the tool can coexist operationally on the system.

7) The tool can be implemented on various hosts.

8) The tool can be ported to various hosts.

9) The tools output can be interchanged between hosts.

Rating Scale

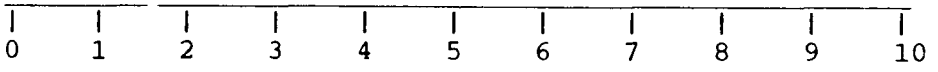


Fault Tolerance

1) The tool has a well defined atomicity of action.

2) When the tool is found to be incorrect, the system can be rolled back to remove the effects of the incorrect actions.

Rating Scale



Self-Instrumented

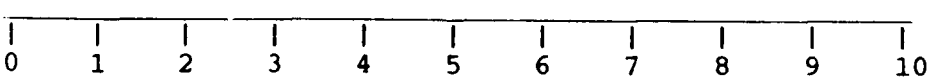
1) The tool contains instrumentation to allow for ease of debugging.

2) Other tools exist for analyzing the results collected by the instrumentation.

3) The tool contains self-test mechanisms to insure that it is working properly.

4) The tool records, maintains, and employs failure records.

Rating Scale



Methodology Support - General

1) The tool supports numerous methodologies to meet the users needs.

2) The tool provides a means to integrate other methodologies.

3) The tool supports all aspects of the methodology employed.

4) When aspects are excluded, the important parts or concepts of the methodology are supported.

5) The tool supports the communication mechanisms of the methodology without alteration.

6) The tool builds in functionality, in addition to the direct support of the methodology, that is useful.

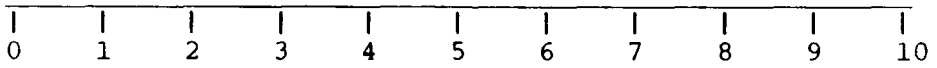
7) The tool is free of functionality that is useless or a hindrance.

8) The tool's flexibility supports the methodology, allowing the user to initially skip or exclude some parts of the methodology and return to it later.

9) The tool provides an adequate scheme to store, organize, and manipulate the products of the application of the methodology.

10) The tool provides guidance to ensure that the concepts of the methodology are followed when using the tool.

Rating Scale



Methodology Support - Specific

1) The tool supports an object oriented methodology.

2) The tool supports a functionally oriented methodology.

3) The tool supports a data oriented methodology.

4) The tool supports real-time structured development.

5) The tool supports entity relationship modeling.

6) The tool supports Petri nets.

7) The tool supports state charts.

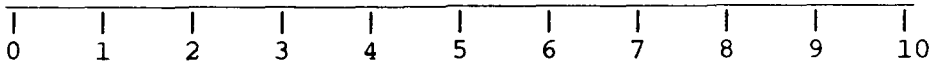
8) The tool supports DARTS.

9) The tool supports ADARTS.

10) The tool supports entity relationship rules.

11) The tool supports automatic data flow balancing.

Rating Scale



Life Cycle Support

1) The tool supports the requirements definition and analysis stage of the software system life cycle.

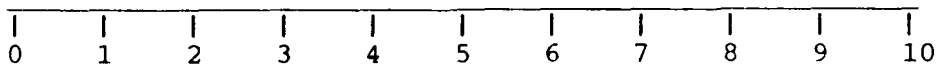
2) The tool supports the design stage of the software system life cycle.

3) The tool supports the implementation stage of the software system life cycle.

4) The tool allows for the generation of Ada code.

5) The tool supports management functions of the software system life cycle.

Rating Scale



Correctness

1) The tool generates output that is consistent with what is dictated by the methodology.

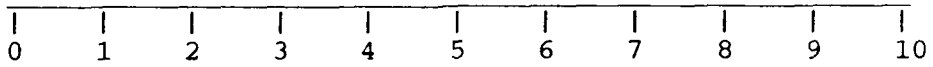
2) The tool checks to see if the methodology is being executed correctly.

3) The tool will not unintentionally or unexpectedly alter data items entered by the user.

4) Outputs generated by the tool are correct by all standards.

5) Transformations generated by the computer will always generate correct results.

Rating Scale



Ease of Use

1) The tool simplifies a problem rather than complicates it.

2) Command names suggest function or graphical symbols representative of function.

3) Commands and command sequences are consistent throughout the system.

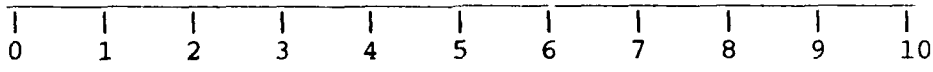
4) The user can do something quickly to see what happens and evaluate results without a long set-up process.

5) Results and produced work products of learning exercises can be disposed of easily without intervention of a second party, i.e. database administrator.

6) The tool provides a small number of functions that allow the user to do the work the tool is intended to do.

7) The tool provides the user with templates or other aids to guide interaction.

Rating Scale



Learnability

1) Prospective tool users require a background necessary to use the tool.

2) A user can use the tool without memorizing an inordinate number of commands.

3) The tool is based on a small number of easy to understand/learn concepts that are clearly explained.

4) The user can learn a small number of simple commands initially, and gradually add more advanced commands as proficiency is developed.

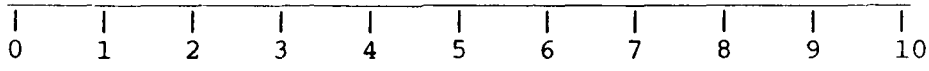
5) The time required to understand and become proficient is acceptable for the average user.

6) A tutorial is provided with the tool.

7) On line help is available with the tool.

8) Expert mode exists for the experienced user.

Rating Scale



Software Engineering Environment

1) The tool is in some ways similar to what AFIT currently does and knows, there is a commonality in the underlying method, process, vocabulary, and notation.

2) The command set is free of conflict with the command set of other tools AFIT uses.

3) The tool runs on the hardware/operating system AFIT currently uses.

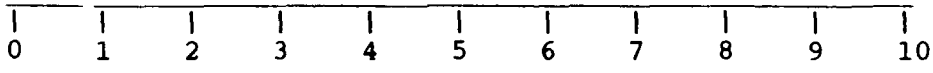
4) Installing the tool is a simple, straightforward process.

5) The tool uses file structures/databases similar to those currently in use at AFIT.

6) The data can be interchanged between the tool and other tools currently employed by AFIT.

7) The tool can be cost-effectively supported by those responsible for maintaining the environment.

Rating Scale

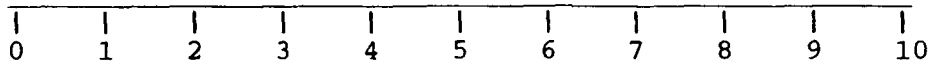


Data Base

1) The tool creates a data base that can be used by other tools.

2) The tool can access an existing central data base.

Rating Scale



Tool History

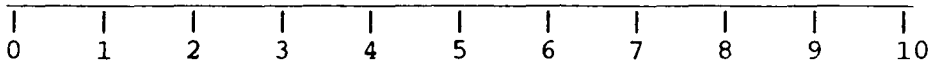
1) The tool has a history that indicates it is sound and mature.

2) The tool has been applied in a relevant application domain.

3) A complete list of all users that have purchased the tool is available.

4) It is possible to obtain evaluations of the tool from a group of users.

Rating Scale



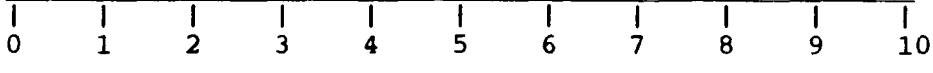
Vendor History

1) There is a staff dedicated to user support.

2) The vendor has a reputation for living up to commitments and promises.

3) Future projections of the company are promising.

Rating Scale



Purchase, Licensing, or Rental Agreement

1) The contract or agreement is explicit enough so that the customer knows what is or is not being acquired.

2) There is a cost reduction for the purchase of multiple copies.

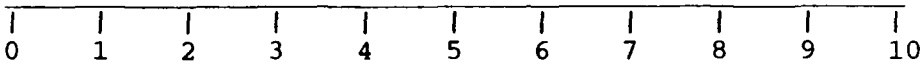
3) A corporate site license is available.

4) The user can return the tool for a full refund during some well defined, reasonable period of time.

5) The user is free of all obligations to the vendor regarding use of objects generated by the tool.

6) The tool is affordable.

Rating Scale



Maintenance Agreement

1) A warranty exists for the tool.

2) The user can purchase a maintenance agreement.

3) The vendor can be held liable for the malfunctioning of the tool.

4) Maintenance agreements will be honored to the customers satisfaction in the case that the vendors sell out.

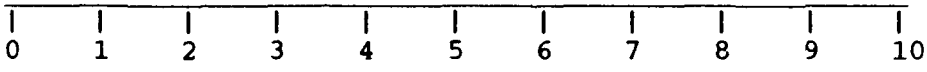
5) The frequency of releases and/or updates to the tool will be reasonable.

6) The maintenance agreement includes copies of releases or updates.

7) The turn around time for problem or bug reports is acceptable.

8) The vendor supports and assists tailoring the tool to the specific user need.

Rating Scale

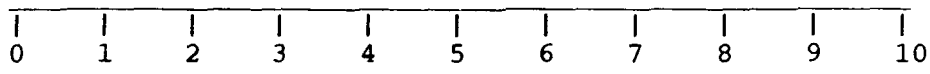


User's Group/User Feedback

1) A user's group exists.

2) The vendor provides a responsive, helpful hot-line service.

Rating Scale



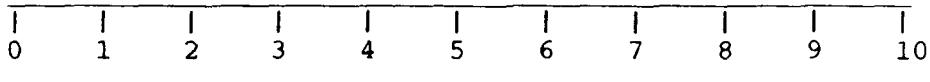
Installation

1) The tool is delivered promptly as a complete package.

2) The vendor provides installation support/consultation.

3) The tool supports both a mainframe and work station environment.

Rating Scale



Training

1) Training is available.

2) Prerequisite knowledge for learning and use of the tool has been defined.

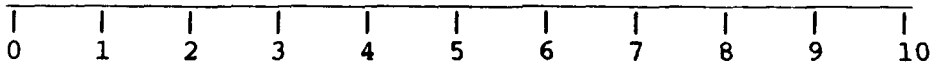
3) Training is customized for AFIT and individuals with attention paid to the needs of different types of users.

4) Training materials or vehicles allow the user to work independently as time permits.

5) The user is provided with examples and exercises.

6) Vendor representatives are product knowledgeable and trained.

Rating Scale



Documentation

1) The tool is supported with installation manuals.

2) The tool is supported with user's manuals.

3) The tool is supported with maintenance manuals.

4) The tool is supported with interface manuals.

5) The documentation provides a description of what a tool does before throwing the user into the details of how to use it.

6) The documentation is readable.

7) The documentation is understandable.

8) The documentation is complete.

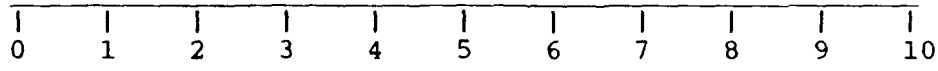
9) The documentation is accurate.

10) The documentation is affordable.

11) The documentation has an indexing scheme to aid the user in finding answers to specific questions.

12) The documentation is promptly and conveniently updated to reflect changes in the implementation of the tool.

Rating Scale



CASE Tool Evaluation (Novice)

Tailoring

1) I could tailor aspects of the tool interface to suit my needs, including application and ability level.

2) I could turn off tool functions that I didn't need.

3) I could create and add new functions to the tool.

4) I could redefine the tool's input and output formats to suit my needs.

Rating

0	1	2	3	4	5	6	7	8	9	10

Intelligence/Helpfulness

1) The tool was interactive.

2) The tool prompted me when it needed a command.

3) The tool checked for command errors.

4) I could control when and how the tool performed its job.

5) The tool gave me quick and meaningful feedback on the system status and the progress of interaction and execution.

Rating

0	1	2	3	4	5	6	7	8	9	10

Graphics

1) The tool provided me the capability to produce hard copy output of diagrams or views I created.

2) The tool provided graphics functions such as copy, move, enlarge/shrink, delete/undelete, and zoom in/out for operations on objects.

3) The tool provided the capability to combine or decompose diagrams I created.

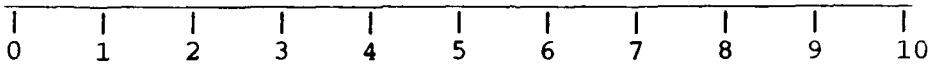
4) The interface was simplified by the use of sound or graphics.

5) The interface included graphical icons with shape and texture.

6) The interface included the ability to color code graphical representations.

7) I could access and retrieve stored information quickly and with little effort while using the tool.

Rating

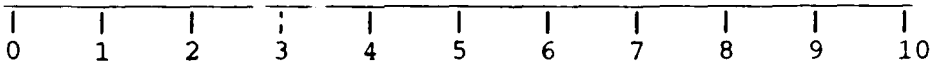


Predictability

1) I usually received the responses from the tool that I expected.

2) It was possible to predict the response of the tool to different types of error conditions.

Rating



Error Handling

1) The tool recovered from errors easily.

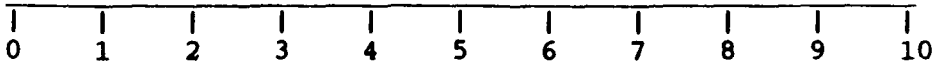
2) The tool had mechanisms that protected me from costly errors.

3) The tool executed periodic saves of my work to ensure that all the work was not lost if a failure occurred during a long session with the tool.

4) The tool protected against damage to its database.

5) The tool helped me correct errors.

Rating



System/Human Interface

1) More than one person at a time was able to use the tool.

2) The tool provided for management of work products for single and multiple users.

3) The tool prevented unauthorized access to or modifications of my data.

4) I found the menus provided easy to use.

5) The tool allowed use of a mouse.

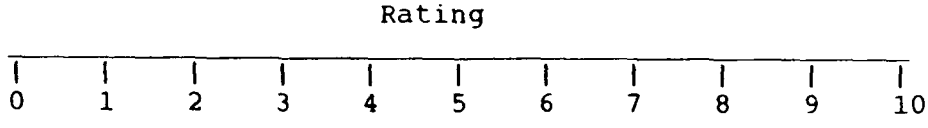
6) The tool allowed use of a keyboard.

7) The tool provided split screen/window capability.

8) The tool provided undo capabilities.

9) I could use external files as input.

10) I could create external files for outside use.



Tool Understanding

1) The tool operated on objects at different levels of abstraction or at different levels of detail.

2) The tool allowed me to modify collections of objects so as to
preserve relationships between them.

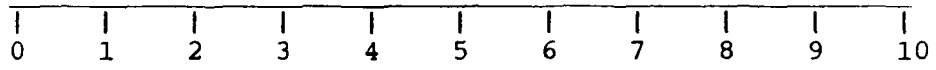
3) The tool allowed me to remove unwanted objects and repair the structure.

4) The tool allowed me to insert objects with proper changes to the structure.

5) The tool performed validation of objects or structures.

6) The tool performed consistency checks between levels of detail.

Rating

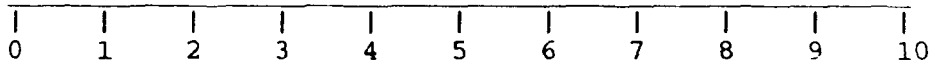


Tool Leverage

1) Global commands could be used on specific object types.

2) The tool applied commands systematically to entire collections of similar objects.

Rating



Tool State

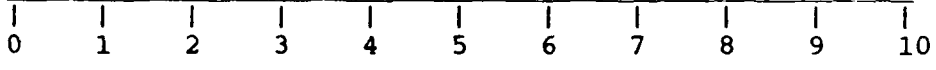
1) The tool provided a macro capability.

2) Macros could be modified.

3) I was able to save the current state of the tool and the objects it was manipulating.

4) I was able to restore the saved state of the tool and the objects it was manipulating.

Rating



Data Dictionary

1) The tool dictionary was automatically updated from the design.

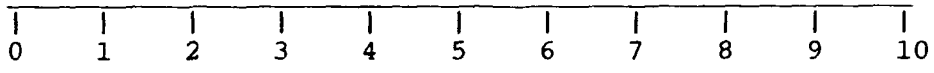
2) The tool automatically validated the dictionary and design.

3) I was able to delete unneeded entries from the data dictionary.

4) I was able to rename data definitions as well as all references to that structure in diagrams and other structures.

5) I was able to create data descriptions to describe in detail in textual form, each data structure and data element.

Rating



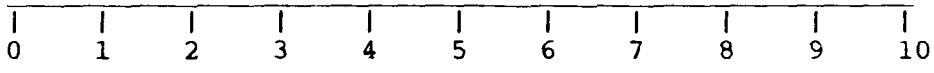
Performance

1) I found the tool's response time acceptable.

2) When the tool ran on the hardware available to me, it was able to handle a development task of the size I required.

3) The tool provided a mechanism to dispose of useless byproducts it creates.

Rating

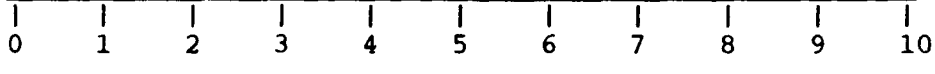


Consistency

1) I could archive and selectively retrieved and access output I generated using the tool.

2) The tool allowed me to undelete and delete objects.

Rating



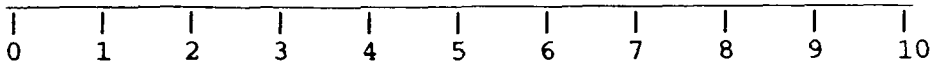
Self-Instrumented

1) The tool contains aids which made debugging easy.

2) The tool contains self-test mechanisms which insured that it worked properly.

3) The tool recorded, maintained, and employed failure records.

Rating



Correctness

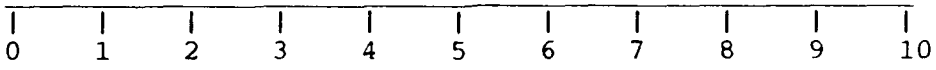
1) The tool generated output that was logically correct.

2) The tool checked to see if the methodology I used was being executed correctly.

3) The tool did not unexpectedly alter data items I entered.

4) Transformations generated by the computer always generated correct results.

Rating



Ease of Use

1) I found that the tool simplified a problem rather than complicated it.

2) I thought the command names or graphical symbols suggested corresponding functions.

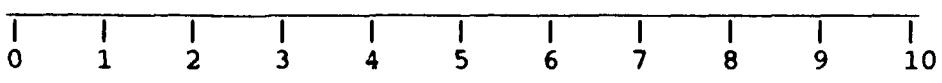
3) I found the commands and command sequences were consistent throughout the system.

4) I could do something quickly to see what happens and evaluate results without a long set-up process.

5) I could easily dispose of results and produced work products of learning exercises without intervention of a second party, i.e. database administrator.

6) The tool provided a small number of functions that allowed me to do the work the tool is intended to do.

Rating



Learnability

1) I was able to use the tool without memorizing an inordinate number of commands.

2) I found the tool was based on a small number of easy to understand/learn concepts that I feel were clearly explained.

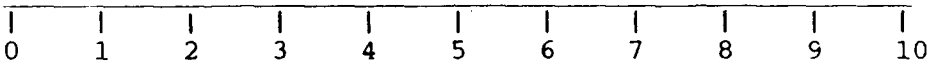
3) I was able learn a small number of simple commands initially, and gradually add more advanced commands as my proficiency with the tool developed.

4) I found time required to understand and become proficient with the tool acceptable.

5) I found the tutorial easy to use and informative.

6) I found the on line help facilities easy to understand and useful.

Rating



Documentation

1) I thought the documentation provided gave a description of what the tool does before throwing me into the details of how to use it.

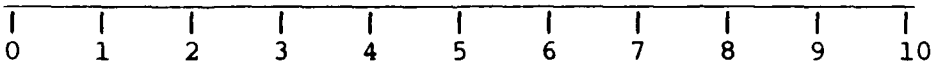
2) I found the provided documentation readable.

3) I found the provided documentation understandable.

4) I found the provided documentation complete.

5) I found the provided documentation accurate.

Rating



Appendix E: Test Cases for CASE Tool Evaluations

This appendix contains a description of the different test cases used by the CASE tool evaluators during the evaluation process.

Test Case 1: Ajax Community College Registration System

During the latter part of a quarter, each student will review the course offerings for the upcoming quarter and make his/her selections. The registration system will review a student's request for courses, verify each course will be offered during the coming quarter and that the student has fulfilled all prerequisite requirements for each course. (These prerequisites are maintained in the Course Catalog database.) If space is still available in the course, the student will be assigned to it and his/her name will be added to the course roster. Each student will receive a schedule listing each course and its time and meeting place. A student roster for each course will be generated for its instructor.

In addition, a summary report is provided to each department listing the enrollment in each of its courses as well as the number of students denied enrollment if the course is "filled". A record is kept of each student denied enrollment so that if a department decides to "open" enrollment (or add another section) those students could be added. A report is also compiled for the President's office which lists the number of students enrolled in each

department. This is used as a basis for allocating TA slots to the departments.

Although in the future (when additional funds become available), we would like to offer an "on line" or "dial-in" registration system which the students could directly access, at the present time we will be satisfied with a system which uses our current mainframe computer and our registration staff. The system must be operational for the fall term of 1992.

Test Case 2: A Spelling Checker

The spelling checker will parse an input document extracting one word at a time. (AS it parses the document, it displays the current document line number and word number on the terminal.) If the word is a "non-word", the spelling checker passes it directly to the output document. Otherwise, the spelling checker consults the main and temporary dictionaries. If either dictionary contains the word, the word is written on the output document.

If neither dictionary contains the word, the spelling checker allows the operator to either:

1 - replace the word with a word typed in from the terminal.

2- add the word to the temporary dictionary.

3 - add the word to the main dictionary.

4 - request the program to display like-spelled words in the main and temporary dictionaries.

The word entered or accepted by the user is written to the output document. The space available in the main and temporary dictionaries will be continuously displayed on the terminal. An error message will be displayed if the user tries to add a word to a full dictionary.

Appendix F: CASE Tool Evaluation Results

The following are the results of the novice and expert evaluations distributed to the students and instructors who evaluated the CASE tools considered. Evaluators were asked to rate a tool based on how well the tool supported each weighted criteria. A rating scale of 0 - 10 was used with a 0 indicating that the tool gave no support to the weighted criteria considered, a 5 indicating that the tool gave moderate support, and a 10 indicating that the tool gave full support.

Criteria Support Rating (Expert)	POSE	DECdesign
Tailoring	5.0	2.0
Intelligence/Helpfulness	6.0	8.0
Graphics - General Capabilities	8.0	7.0
Graphics - Specific Capabilities	6.0	4.0
Predictability	9.0	9.0
Error Handling	8.0	6.0
System/Human Interface	4.0	9.0
Tool Understanding	6.0	9.0
Tool Leverage	5.0	8.0
Tool State	4.0	6.0
Data Dictionary	5.0	8.0
Integration	3.0	9.0
Performance	5.0	5.0
Consistency	6.0	8.0
Evolution	8.0	6.0

Criteria Support Rating (Expert)	POSE	DECdesign
Fault Tolerance	3.0	8.0
Self-Instrumented	1.0	3.0
Methodology Support - General	7.0	9.0
Methodology Support - Specific	7.0	6.0
Life Cycle Support	6.0	4.0
Correctness	6.0	9.0
Ease of Use	8.0	9.0
Learnability	8.0	7.0
Software Engineering Environment	4.0	8.0
Data Base	0.0	8.0
Tool History	5.0	5.0
Vendor History	10.0	9.5
Purchase, Licensing, or Rental Agreement	7.0	10.0
Maintenance Agreement	7.0	10.0
User's Group/User Feedback	5.0	9.0
Installation	7.0	10.0
Training	8.0	9.0
Documentation	8.0	3.0

Criteria Ratings (Novice Evaluations)	POSE	DECdesign	ObjectMaker
Tailoring	3.1	6.0	6.0
Intelligence/Helpfulness	5.7	9.0	6.0
Graphics	5.6	8.0	7.0
Predictability	6.0	8.0	7.0
Error Handling	2.7	5.5	5.0

Criteria Ratings (Novice Evaluations)	POSE	DECdesign	ObjectMaker
System/Human Interface	3.6	9.5	7.0
Tool Understanding	5.1	9.5	7.0
Tool Leverage	5.3	9.0	8.0
Tool State	6.6	7.0	6.0
Data Dictionary	3.1	7.5	7.0
Performance	6.4	9.0	8.0
Consistency	2.7	6.0	9.0
Self-Instrumented	4.6	9.0	3.0
Correctness	5.0	7.5	8.0
Ease of Use	6.1	9.5	8.0
Learnability	6.9	8.5	9.0
Documentation	3.7	5.5	7.0

Appendix G: CASE Tool Generated Reports

The following diagrams were generated using the CASE tools evaluated as part of the thesis effort. Test case 2 was used as a basis for the designs.

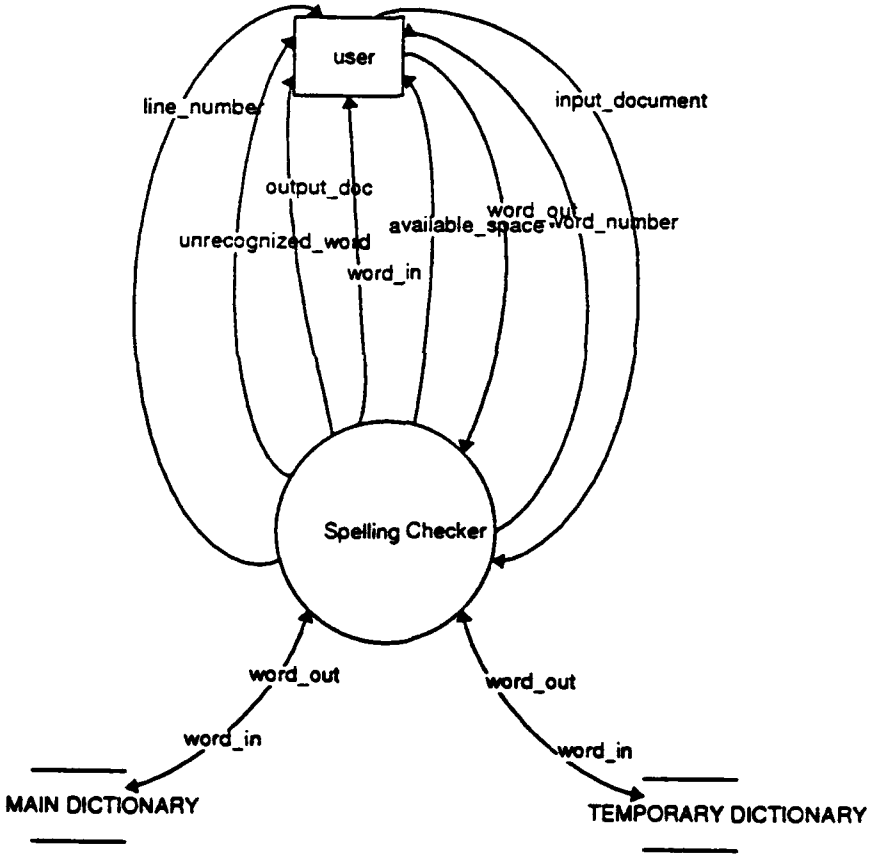
The first set of diagrams were created using DECdesign. Yordon's methodology was utilized to create a context diagram and data flow diagrams. These are located on pages G.2 through G.6. Also generated are an entity relationship diagram, page G.7, and a structure chart, page G.8.

The second set of diagrams were generated using POSE and are the same type of diagrams which were generated using DECdesign. The context diagram and DFDs are located on pages G.9 through G.11. An entity relationship diagram, page G.14, and a structure chart, G.15, were also generated.

The final set of diagrams were generated using ObjectMaker. The context diagram and DFDs are located on pages G.16 through G.20. An entity relationship diagram was also generated on page G.21.

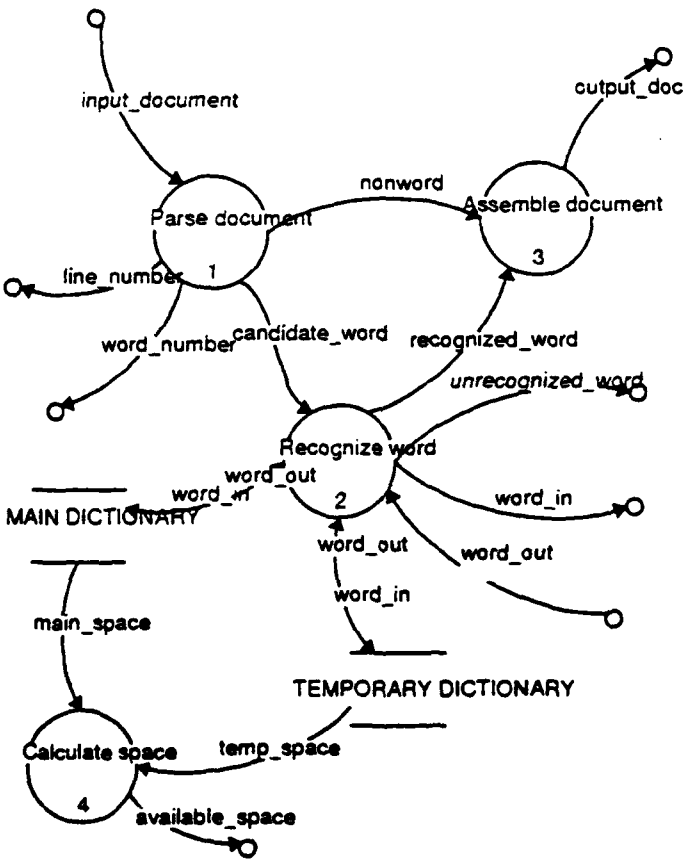
Context Diagram

Label: **Spell_Checker**
Structure Chart: No associated data
Predefined Attributes:
Creation Date: 10-JUL-1991 16:12:21.83
Modification Date: 9-AUG-1991 14:31:35.00



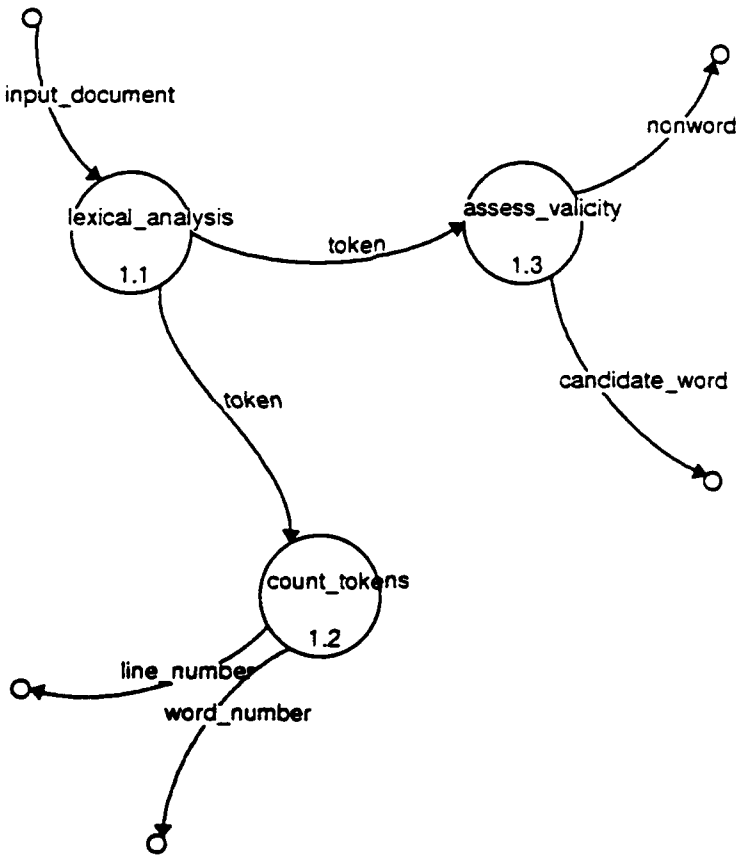
Data Flow Diagram

Label: **Spelling Checker**
Structure Chart: No associated data
Predefined Attributes:
Creation Date: 8-AUG-1991 09:25:48.25
Modification Date: 12-AUG-1991 09:26:17.31



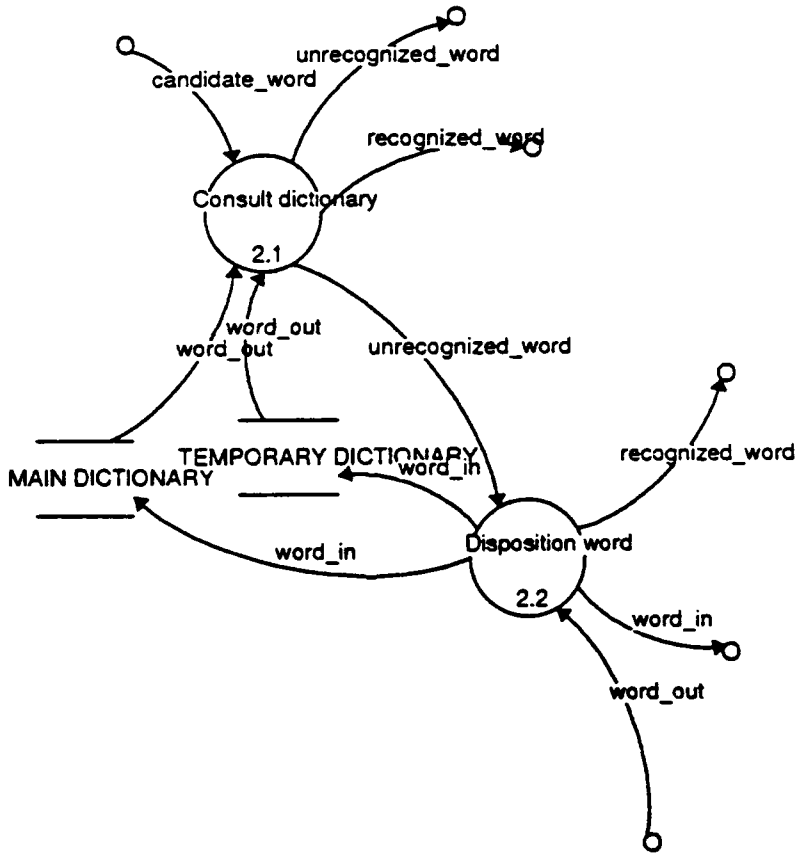
Data Flow Diagram

Label: **Parse document**
Sequence Number: 1
Structure Chart: No associated data
Predefined Attributes:
Creation Date: 8-AUG-1991 09:51:20.81
Modification Date: 12-AUG-1991 09:33:44.76



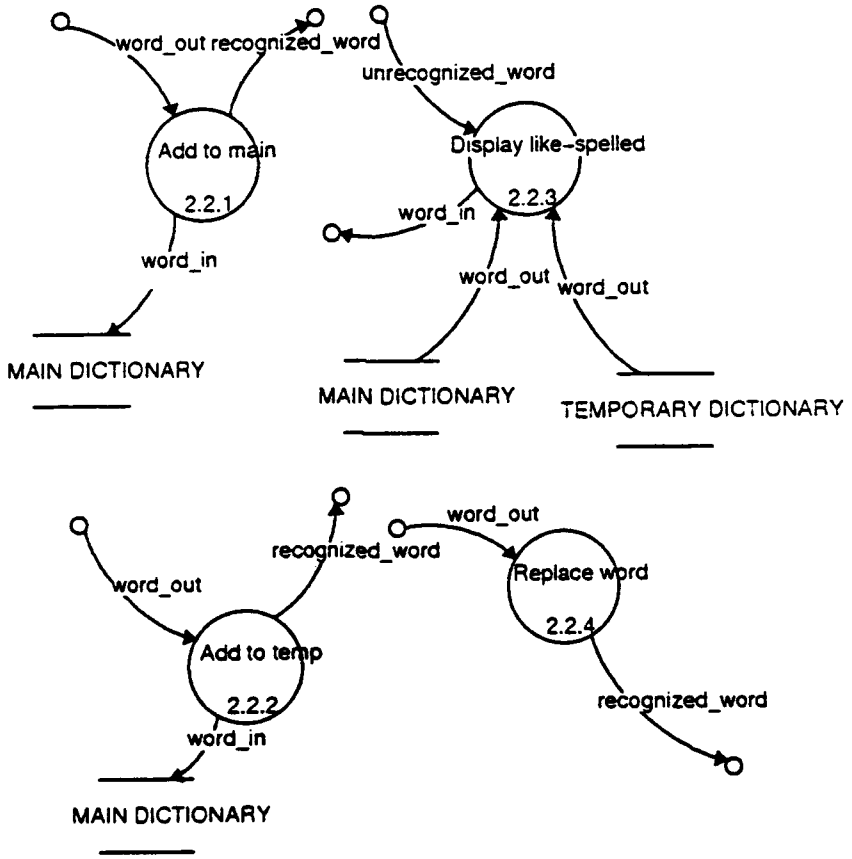
Data Flow Diagram

Label: **Recognize word**
Sequence Number: 2
Structure Chart: No associated data
Predefined Attributes:
Creation Date: 8-AUG-1991 10:09:22.20
Modification Date: 12-AUG-1991 09:39:25.49



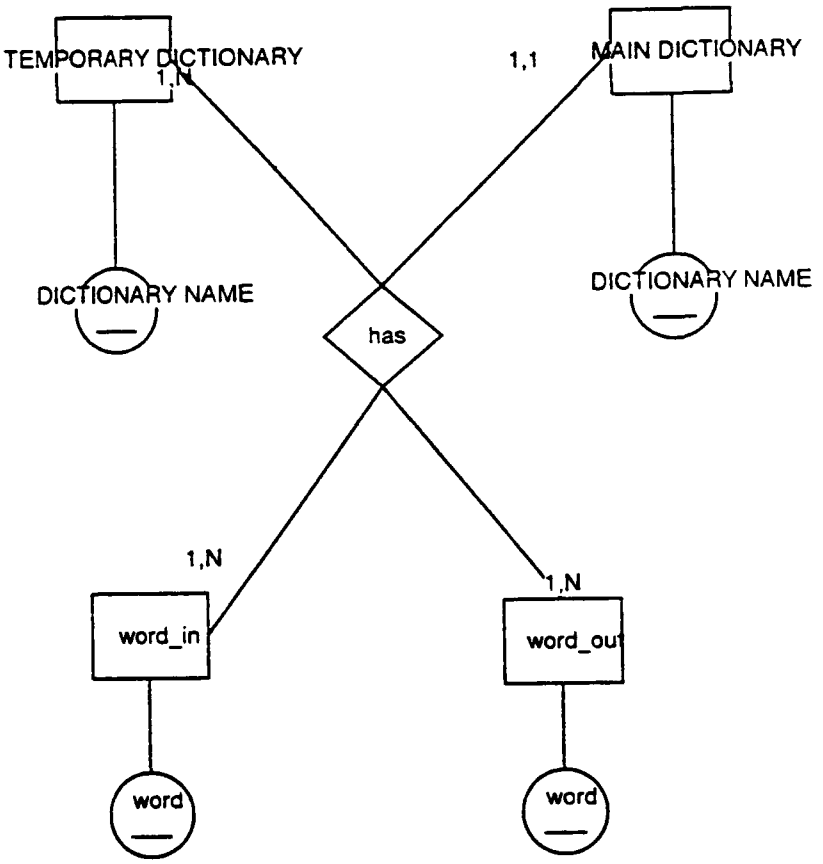
Data Flow Diagram

Label: **Disposition word**
Sequence Number: 2.2
Structure Chart: No associated data
Predefined Attributes:
Creation Date: 8-AUG-1991 10:49:59.26
Modification Date: 12-AUG-1991 10:20:43.25



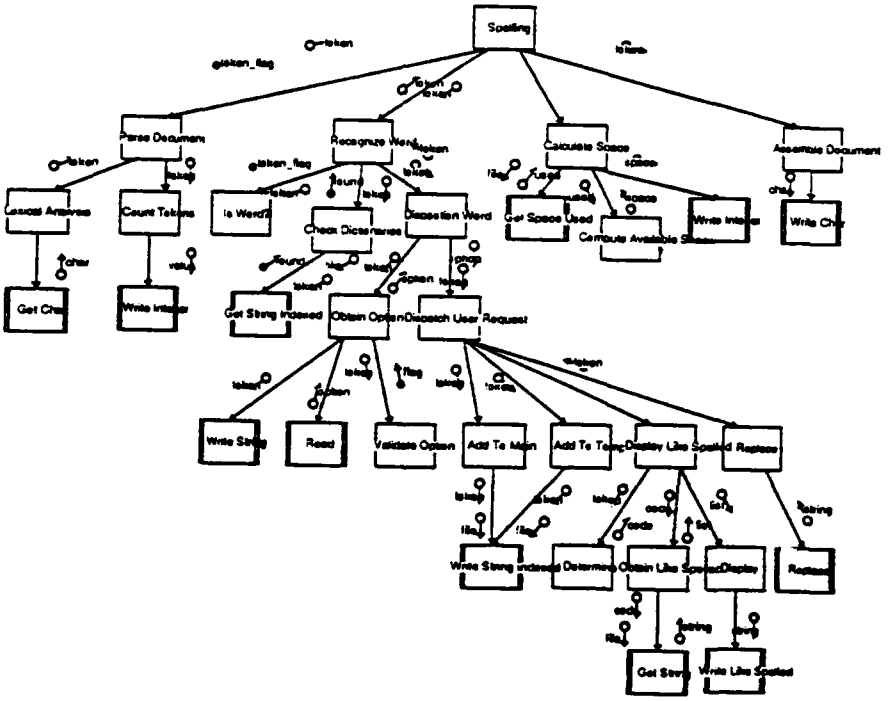
Extended Entity Relationship Diagram

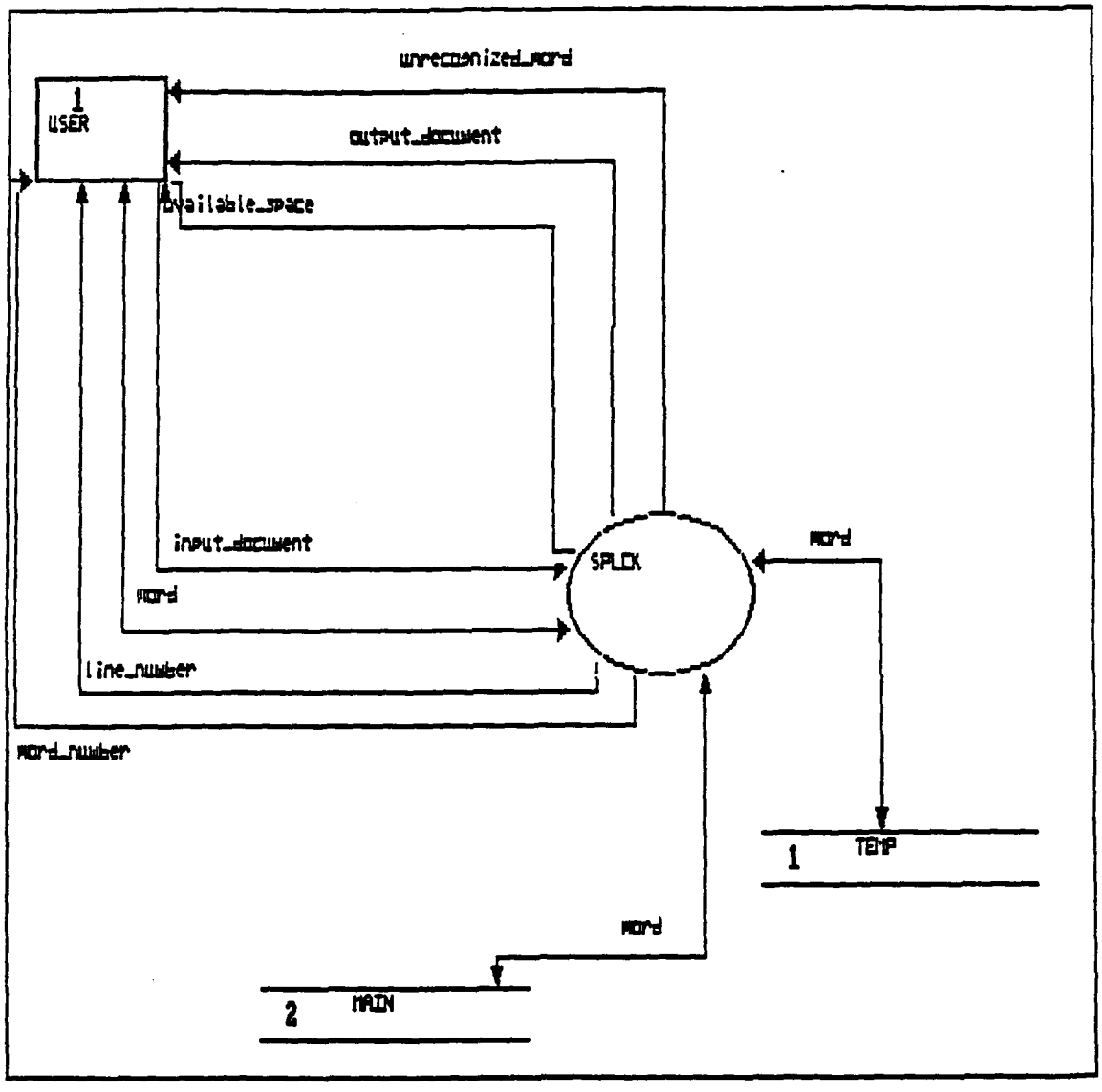
Label: **unnamed**
Predefined Attributes:
Creation Date: 13-AUG-1991 13:33:40.85
Modification Date: 14-AUG-1991 10:52:47.73



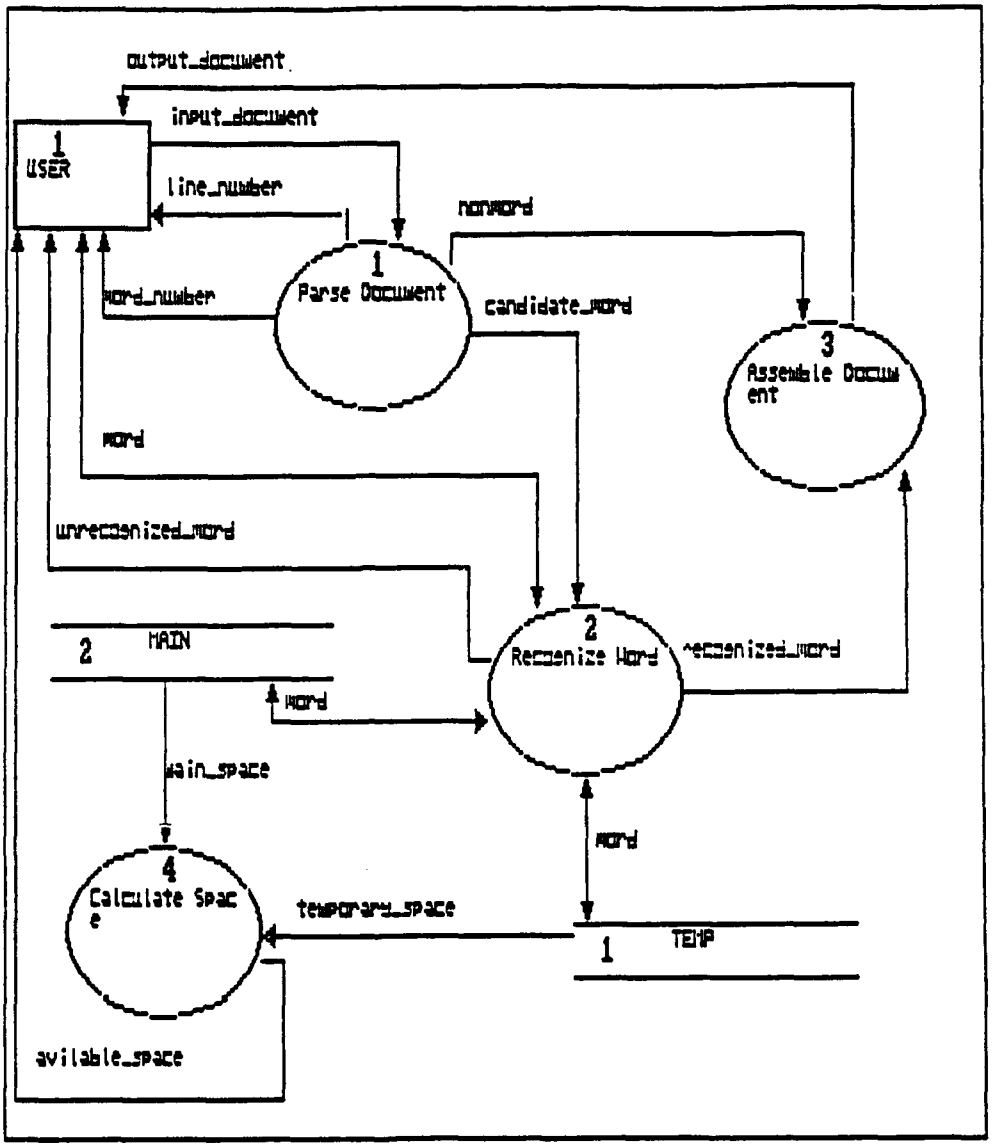
Structure Chart

Label: **Spelling SC**
Predefined Attributes:
Creation Date: 14-AUG-1991 11:01:08.12
Modification Date: 19-AUG-1991 09:40:23.07

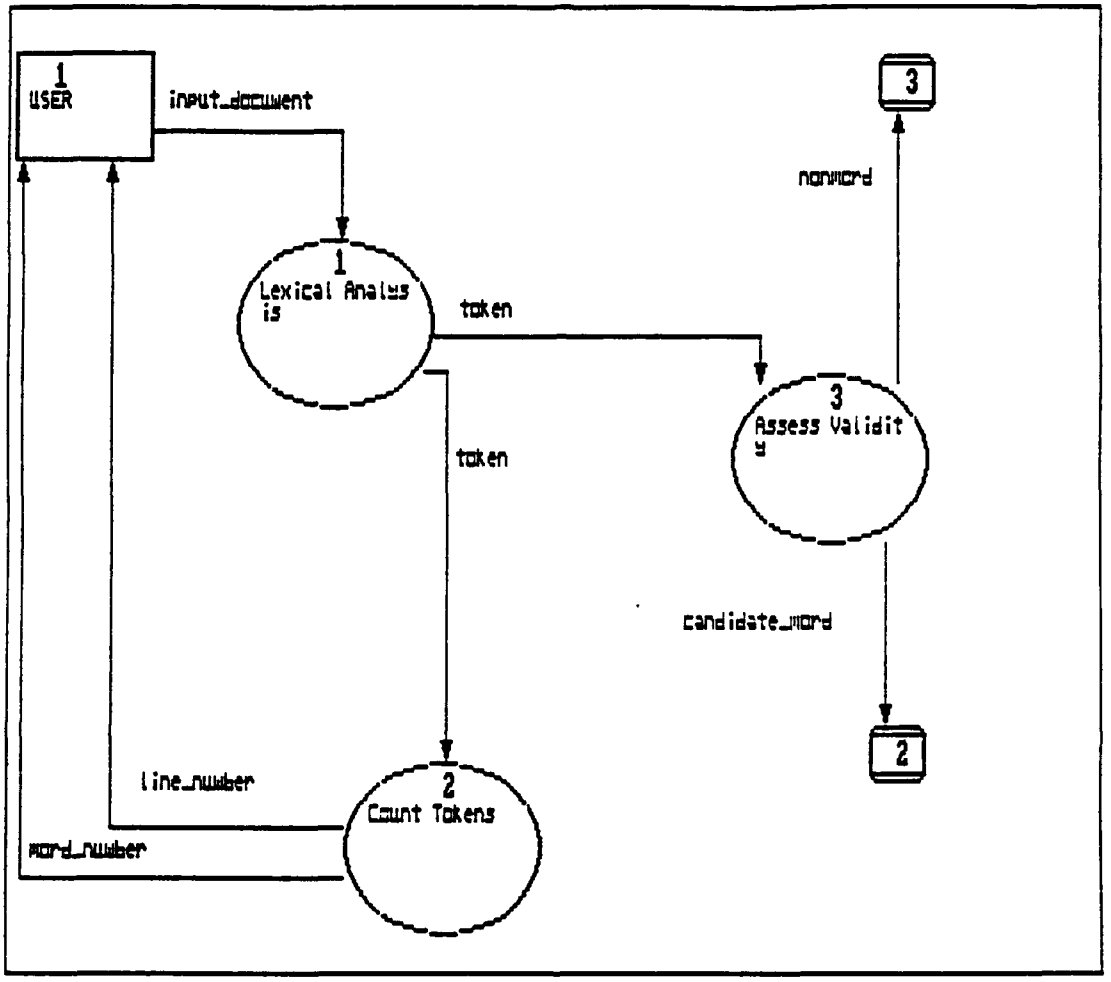




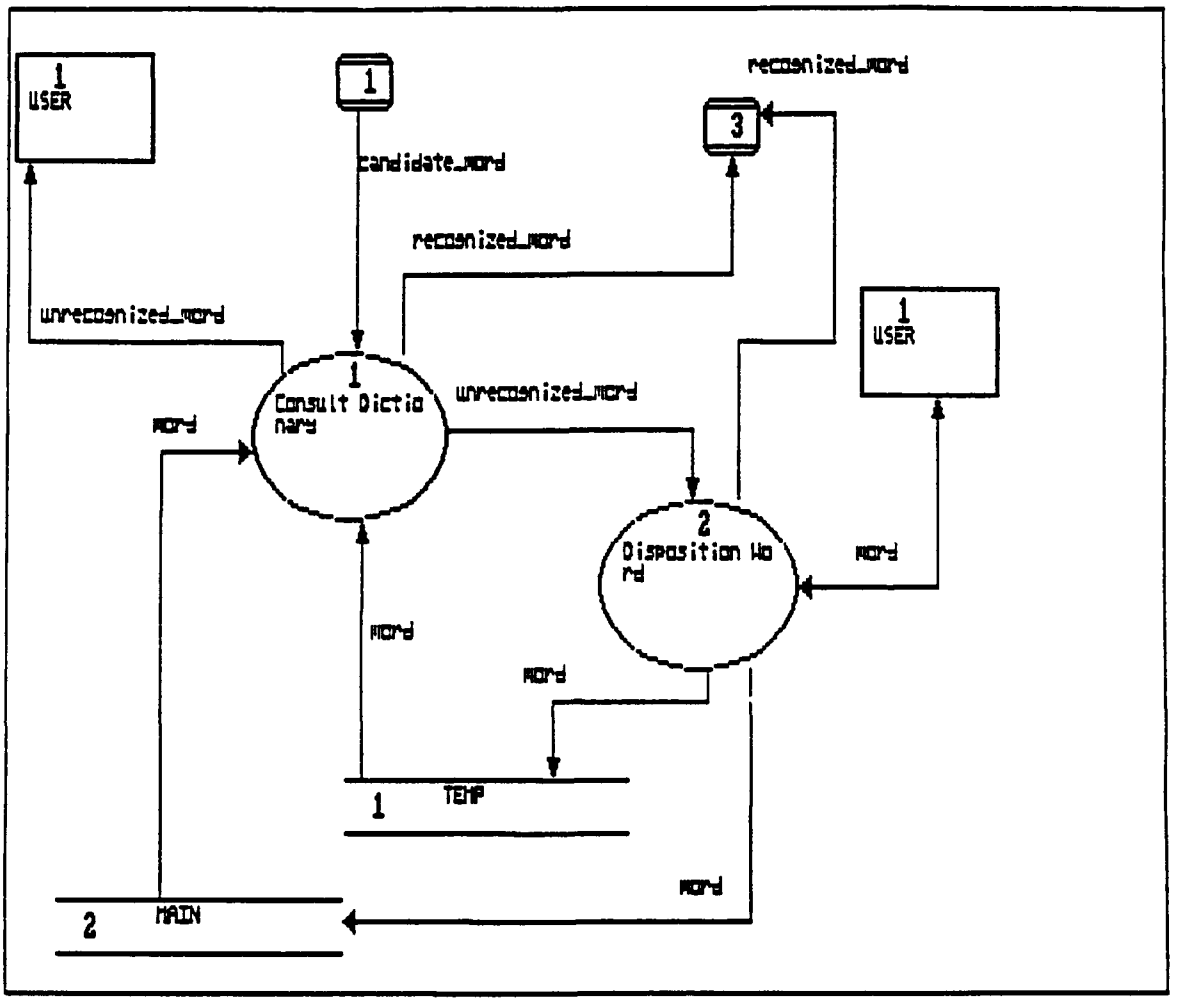
Context Id : SPELL Level : Context



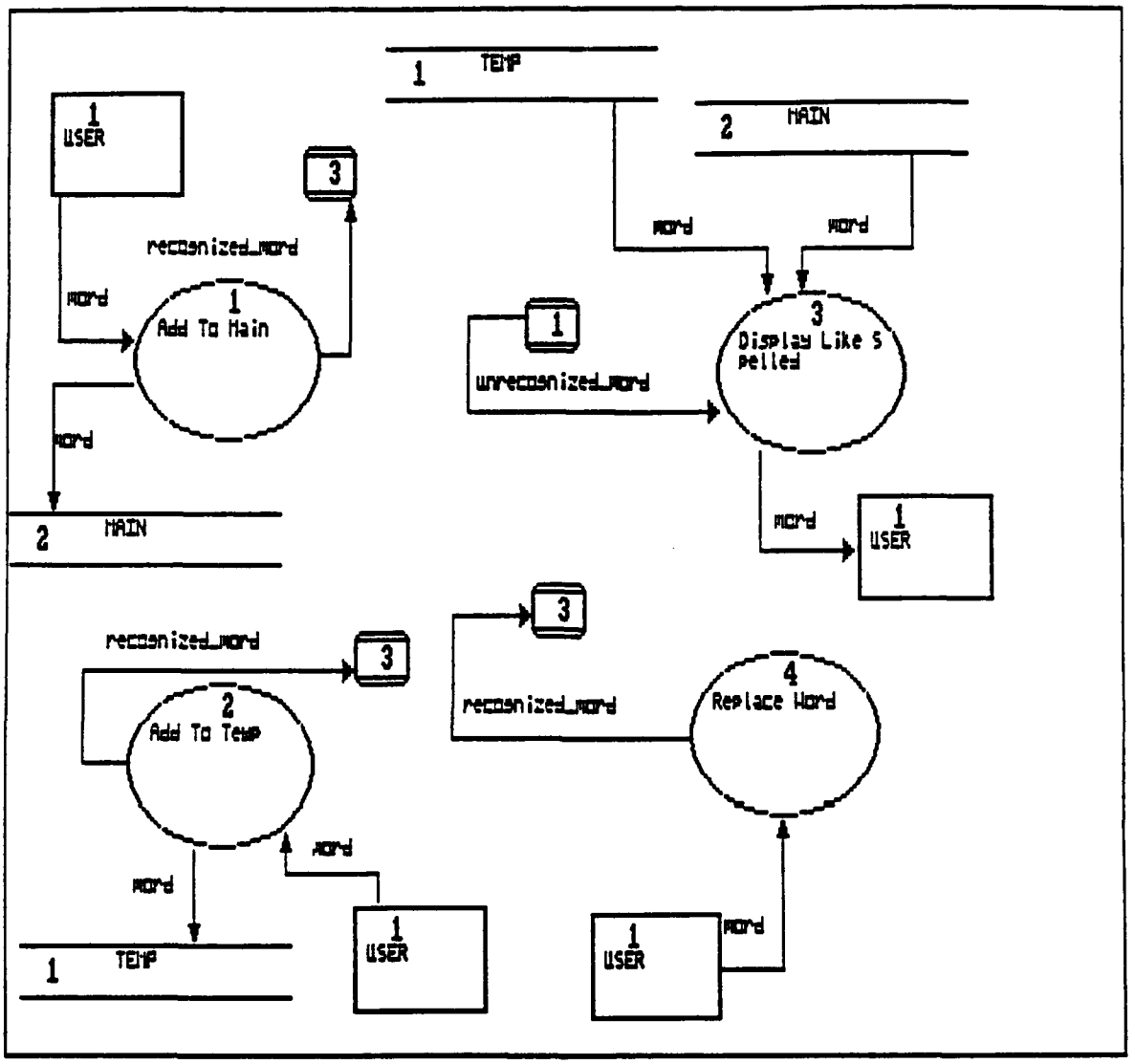
Context Id : SPELL Level : 1



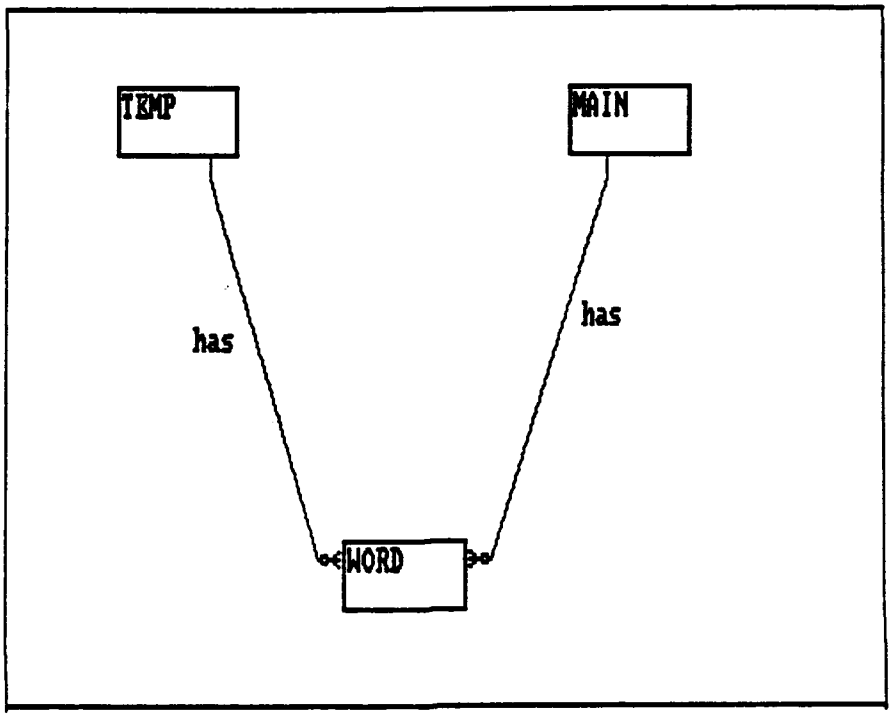
Context Id : SPELL Level : 1.1

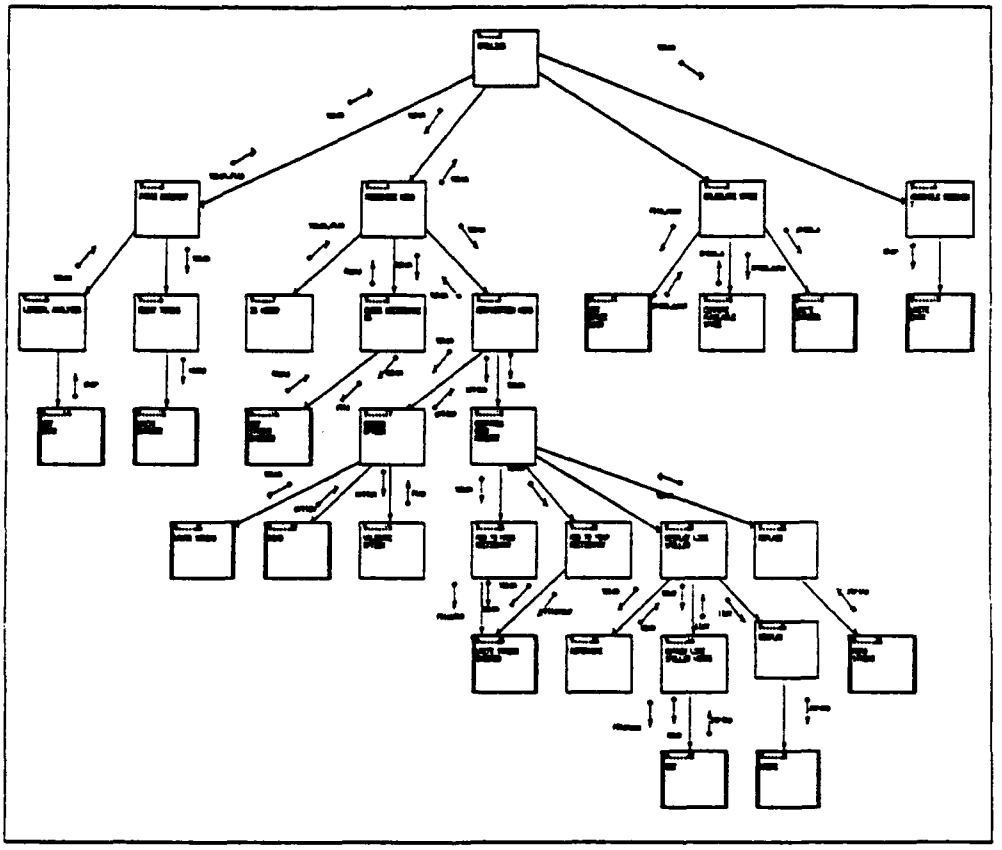


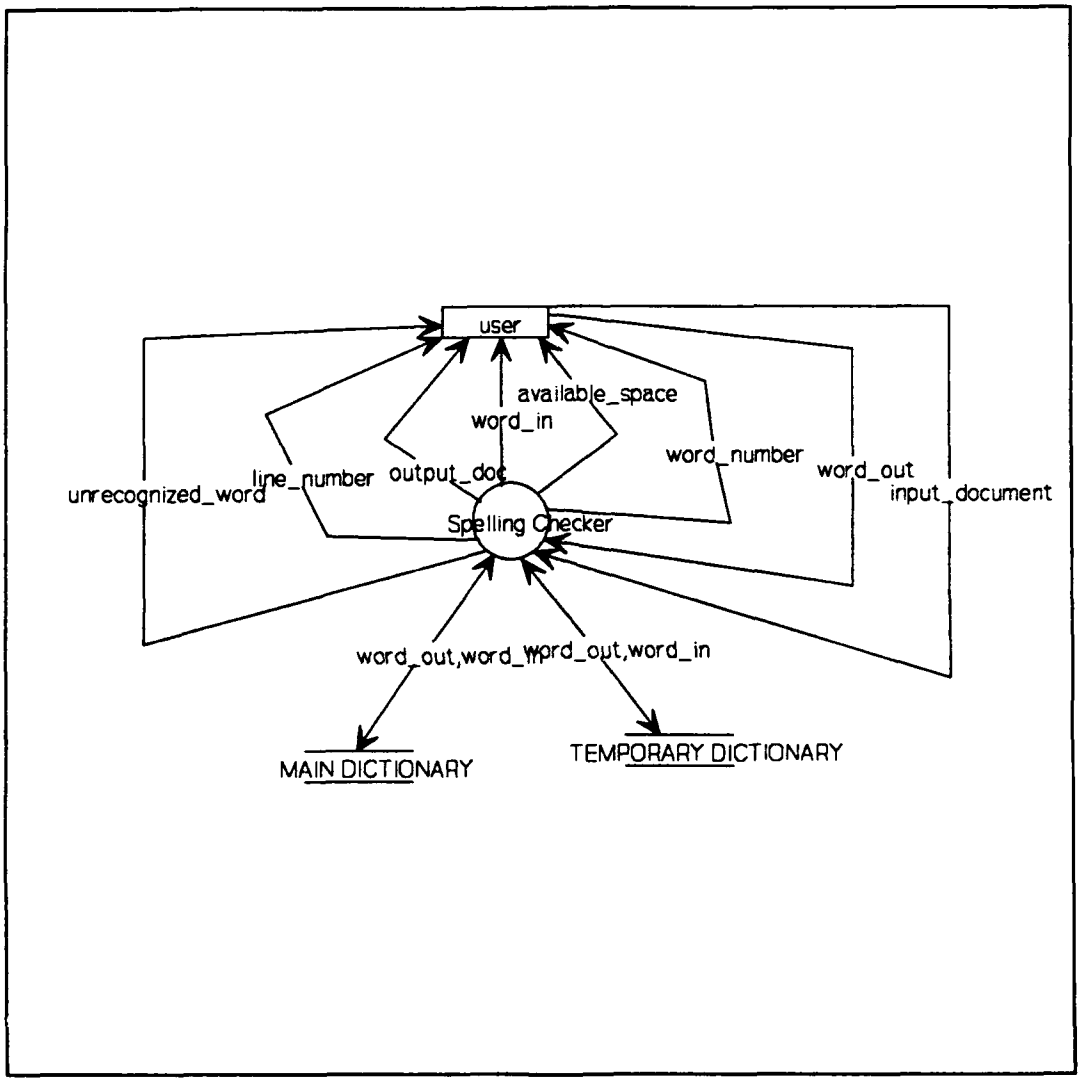
Context Id : SPELL Level : 1.2



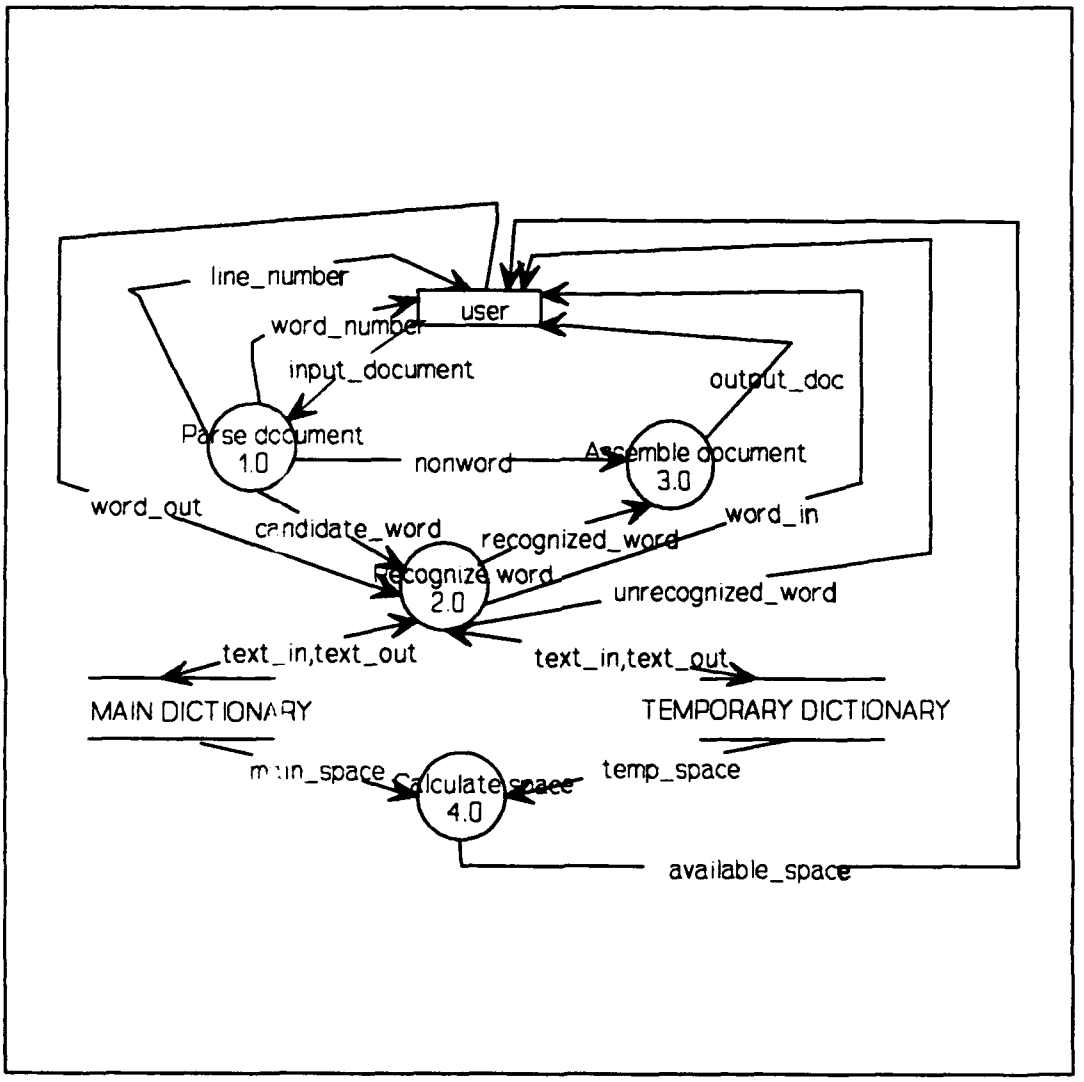
Context Id : SPELL Level : 1.2.2


TEMP
MAIN
has
has
WORD

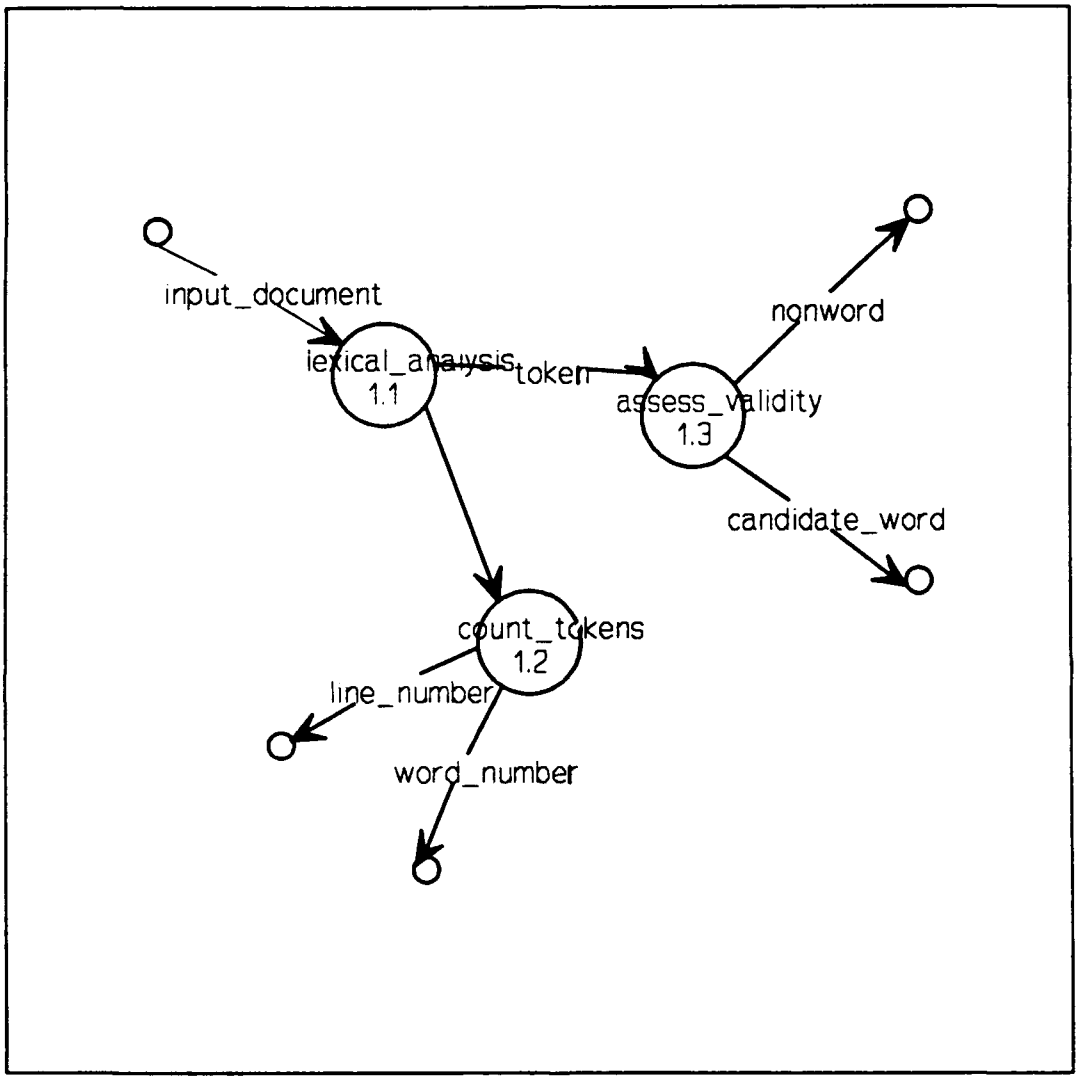




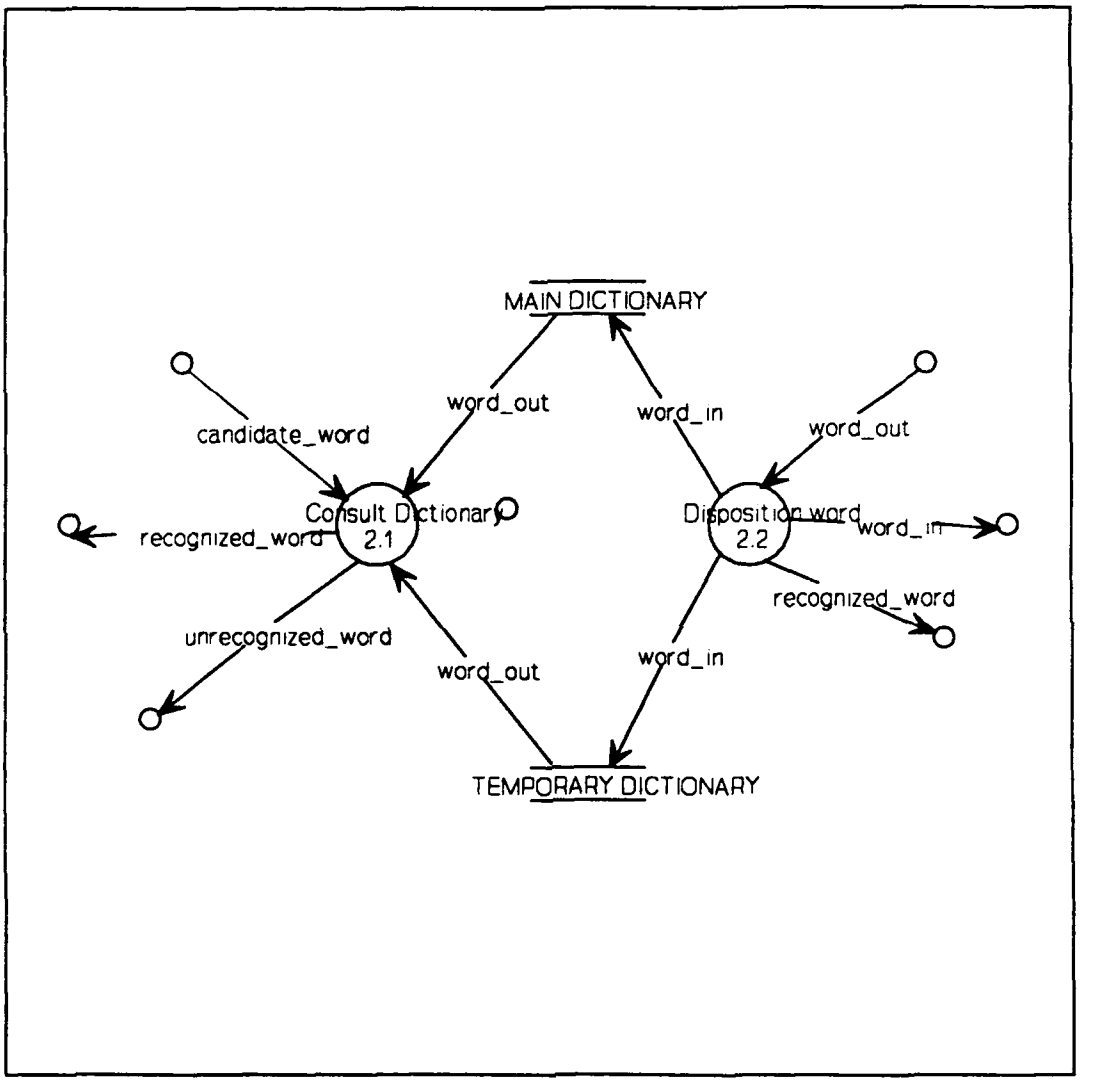
Spelling Checker Context Diagram (ObjectMaker)



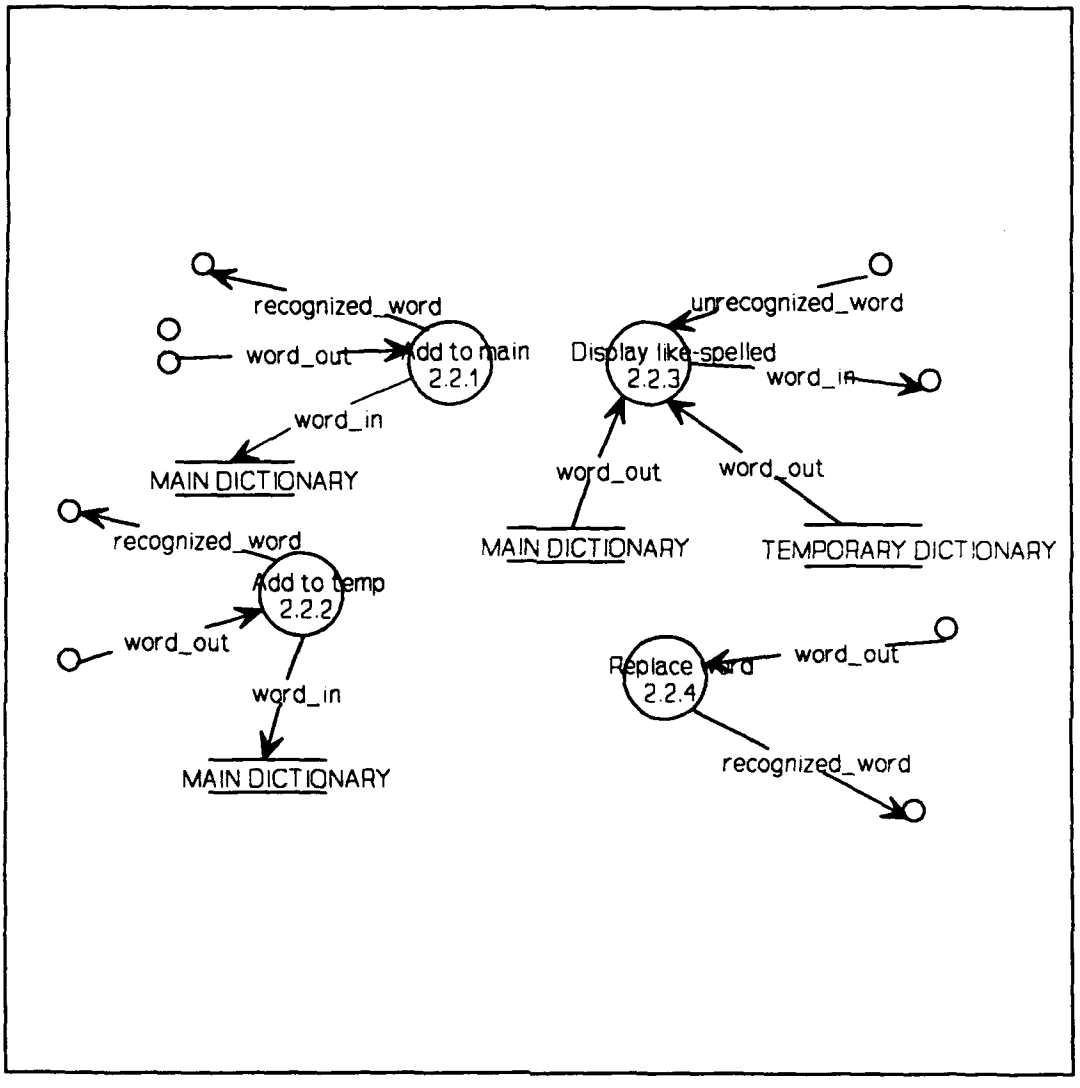
DFD Level 1 (ObjectMaker)



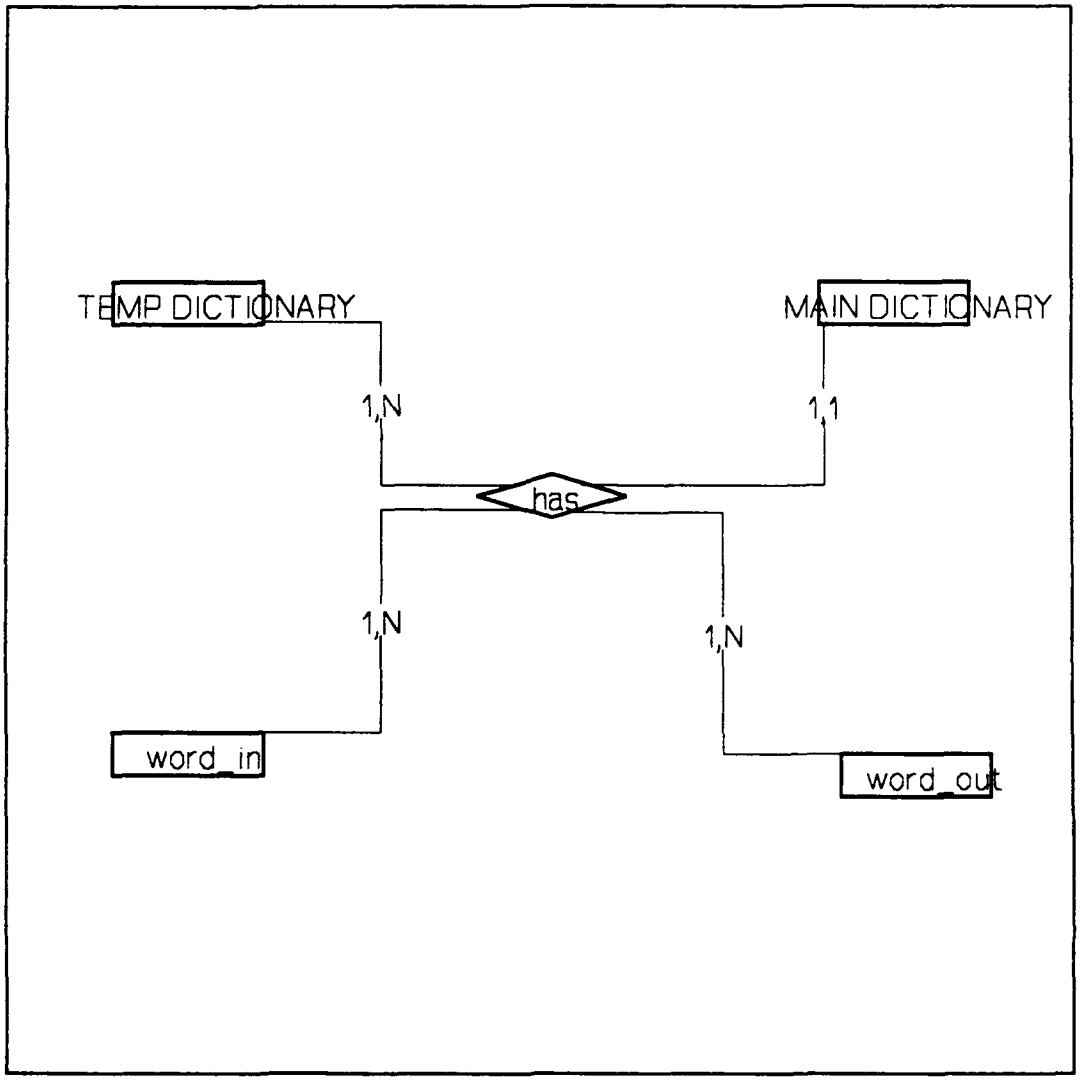
DFD Level 1.1 (ObjectMaker)



DFD Level 2.0 (ObjectMaker)



DFD Level 2.3 (ObjectMaker)



Entity Relationship Diagram (ObjectMaker)

Bibliography

1. Air Force Software Management Group (HQ USAF/SCW). Air Force Software Management Plan, Version 1.0, 1990.

2. Anderson, E.E., "A Heuristic for Software Evaluation and Selection." Software Practice and Experience, 19-8: 707-717 (August 1989).

3. Bailor, Paul D. Class handout distributed in CSCE 595, Software Generation and Maintenance. School of Engineering, AFIT, Wright-Patterson AFB, OH, May 1991.

4. Baram, G. and G. Steinberg. "Selection Criteria for Analysis and Design CASE Tools," ACM SIGSOFT Software Engineering Notes, 14-6: 73-80 (October 1989).

5. Bisiani, R., F. Lecouat, and V. Ambriola. "A Tool to Coordinate Tools," IEEE Software: 17-25 (November 1988).

6. Bruce, T. A., J. Fuller, and T. Moriarty, "So You Want a Repository," Database Programming & Design: 60-69 (May 1989).

7. Burkhard, D. L., "Implementing CASE Tools," Journal of Systems Management, 40-5: 20-28 (May 1989).

8. Buxton, J. N. and L. E. Druffel, "Requirements for an Ada Programming Support Environment: Rationale for STONEMAN," Software Engineering Environments, edited by H. Hunke, North Holland Publishing Company, 1981.

9. Charette, R. N., Software Engineering Environments: Concepts and Technology. New York: Intertext Publications, 1986.

10. Chikofsky, E. J., "How to Lose Productivity With Productivity Tools," Productivity: Progress, Prospects, and Payoff: 1-4 (June 1988).

11. Christoph, B. A.,"A Practical Approach to the Selection of Software Development Tools," Proceedings of the 18th Hawaii International Conference on Systems Science. 524-553. 1985.

12. Coallier, F., "A Strategy for CASE Tool Standard," IEEE Ninth Annual International Phoenix Conference on Computers and Communications. 380-384. 1990.

13. Danziger, M. R. and P. S. Haynes, "Managing the CASE Environment," Journal of Systems Management, 40-5: 29-32 (May 1989).

14. Dart, S. A., R. J. Ellison, P. H. Feiler, and A. N. Habermann, "Software Development Environments," IEEE Computer: 18-28 (November 1987).

15. Department of Defense, Requirements for Ada Programming Support Environments, STONEMAN, version 1.0: (February 1980).

16. Emrick, R. D., "Considering Computer Resource Consumption in the Selection of Appropriate Development Software," EDP Performance Review, 13-11: 1-6 (November 1985).

17. Firth, R. et al. A Guide to the Classification and Assessment of Software Engineering Tools, SEI Institute Report 87-TR-10, Carnegie-Mellon University, 1987.

18. Forte, G., "In Search of the Integrated CASE Environment," CASE OUTLOOK, 3-2: 5-12 (1989).

19. Frewin, G. D., "Metrics in Procurement - a Discussion Paper," Measurement for Software Control and Assurance, Elsevier Applied Science: 89-102 (1989).

20. Gibson, M., "A Guide to Selecting CASE Tools," Datamation: 65-66 (July 1988).

21. Gibson, M. L., C. A. Synder, and R. K. Rainer, Jr., "CASE: Clarifying Common Misconceptions," Journal of Systems Management, 40-5: 12-19 (May 1989).

22. Glickman, S. and M. Becker, "A Methodology for Evaluating Software Tools," Conference on Software Tools. 190-199. 1985.

23. Glass, R. L., "Recommended: A Minimum Standard Toolset," ACM SIGSOFT Software Engineering Notes, 7-4: 3-11 (October 1982).

24. Hartrum, T. C., et. al., "Evaluating User Satisfaction of an Interactive Computer Program," Proceedings of the IEEE 1989 National Aerospace & Electrical Conference, 2: 508-514 (1989).

25. Hatley, D. J., "Parallel System Development: A Reference Model for CASE Tools," IEEE Ninth Annual International Phoenix Conference on Computers and Communications. 364-372. 1990.

26. Henderson, P. B. and D. Notkin, "Integrated Design and Programming Environment," IEEE Computer: 12-16 (November 1987).

27. Hoffnagle, G. F. and W. E. Beregi, "Automating the Software Development Process." IBM Systems Journal, 24-2: 102-120 (1985).

28. Houghton, R. C., Jr. and D. R. Wallace, "Characteristics and Functions of Software Engineering Environments: An Overview," ACM SIGSOFT Software Engineering Notes, 12-1: 64-84 (January 1987).

29. Humphrey, W. S., Managing the Software Process. United States: Addison-Wesley Publishing Company Inc, 1990.

30. Humphrey, W. S. CASE Planning and the Software Process, SEI Institute Report 89-TR-26, Carnegie-Mellon University, 1989.

31. Jones, C., "The Cost and Value of CASE," CASE OUTLOOK, 1-4: 1-15 (1987).

32. Kemerer, C. F., "An Agenda For Research in the Managerial Evaluation of Computer-Aided Software Engineering (CASE) Tool Impacts," Proceedings of the 22nd Annual Hawaii International Conference on Systems Science. 219-228. January 1989.

33. Knight, R., "CASE Paybacks Perceived, if Not Exactly Measured," SOFTWARE NEWS: 56-64 (February 1987).

34. Lawlis, P. K., Supporting Selection Decisions Based on The Technical Evaluation of Ada Environments and Their Components, PhD dissertation. Arizona State University, August 1989.

35. Lehman, M. M. and W. M. Turski, "Essential properties of IPSEs," ACM SIGSOFT Software Engineering Notes, 12-1: 52-55 (January 1987).

36. Lemmp, P., "Development and Project Management Support With the Integrated Software Engineering Environment, EPOS," Software Engineering Environments, edited by I. Sommerville, London: Peter Peregrinus Ltd, 1984.

37. Lempp, P., and R. Lauber, "What Productivity Increases to Expect From a CASE Environment: Results of a User Survey," Productivity: Progress, Prospects, and Payoff: 13-19 (June 1988).

38. Lyon, L., "CASE and the Database," Database Programming & Design: 28-32 (May 1989).

39. Marmelstein, R. E., "Guidelines for Evaluation of Software Engineering Tools," Software Engineering and its Application to Avionics: 17.1-17.6 (1988).

40. Martin, J. and C. McClure, "Buying Software off the Rack," Harvard Business Review, 61-6: 32-52 (1988).

41. Martin, R., "Evaluation of Current Software Costing Tools," ACM SIGSOFT Software Engineering Notes, 13-3: 49-51 (July 1988).

42. Matthews, E. and G. Burns, "VADS APSE: An Integrated Ada Programming Support Environment," SETA1 Conference. April 1990.

43. McKay, C. W., "A Proposed Framework for the Tools and Rules to Support the Life Cycle of the Space Station Program," Proceedings of the IEEE Compass '87 Conference. June 1987.

44. Mosley D., "Breaking Down the Barriers to CASE," American Programmer, 2-9: 11-17 (1990).

45. Necco, C. R., N. W. Tsai, and K. W. Holgerson, "Current Usage of CASE Software," Journal of Systems Management, 40-5: 6-11 (May 1989).

46. Norman, R. J. et al., "CASE Technology Transfer: A Case Study of Unsucessful Change," Journal of Systems Management, 40-5: 33-41 (May 1989).

47. Norman, R. J. and J. F. Nunamaker, Jr., "CASE Productivity Perceptions of Software Engineering Professional," Communications of the ACM, 32-9: 1102-1108 (September 1989).

48. Oestreich, H., "Classification, Evaluation and Selection of Tools," COMPAS '84, Computer Applications, Software Systems: 289-303 (1984).

49. Osterweil, L., "Software Environment Research: Directions for the Next Five Years," IEEE Computer: 35-43 (1981).

50. Penedo, M. H., W. H. Riddle, "Guest Editors' Introduction Software Engineering Environment Architectures," IEEE Transactions on Software Engineering, 14-6: (June 1988).

51. Perry, D. E. and G. E. Kaiser, "Models of Software Development Environments," Proceedings of the 10th International Conference on Software Engineering. 60-68. April 1988.

52. Peterson, G., J. Van Buren, S. Nilson. Requirements Analysis and Design Tools: Interim Report, 2 November 1990.

53. Pressman, R. S., "Selecting and Justifying CASE Tools," CASE OUTLOOK, 1-6: 1-11 (1987).

54. Radding, P. L., "Achieving High Quality Systems: Making CASE Work in Your Orginization," IEEE Ninth Annual International Phoenix Conference on Computers and Communications. 356-363. 1990.

55. Salyers, W., Sales Representative. Telephone interview. Rational, Dayton OH, 30 July 1991.

56. Scheffer, P. A., A. H. Stone, III, W. E. Rzepka, "A CASE Study of SREM," Computer: 47-54 (April 1985).

57. Sharon, D., "Smart CASE Shopping," IEEE Ninth Annual International Phoenix Conference on Computers and Communications. 376-379. 1990.

58. Smith, D., "Evaluating and Selecting CASE Tools," Software Engineering: 22-29 (September/October 1990).

59. Snizek, W., Sales Representative. Telephone interview. Rational, Dayton OH, 29 July 1991.

60. Stinson, W., "Views of Software Development Environments: Automation of Engineering and Engineering of Automation," ACM SIGSOFT Software Engineering Notes, 14-6: 32-41 (October 1989).

61. Steubing, H. G., "A Software Engineering Environment (SEE) for Weapon System Software," IEEE Transactions on Software Engineering, SE-10-4: 384-397 (July 1984).

62. Troy, D. A., "An Evaluation of CASE Tools," Proceedings of COMPSAC 87. 124-30. 1987.

63. Ward, P. T., "Embedded Behavior in Pattern Languages: A Contribution to a Taxonomy of CASE Languages," The Journal of Systems and Software, 9: 109-128 (1989).

64. Weiderman, N. H., A. N. Habermann, M. W. Borger, and M. H. Klein, "A Methodology for Evaluating Environments," ACM SIGPLAN Notices, 22-1: 199-207 (1987).

65. Wilson, D. M., "CASE: guidelines for success," Information and Software Technology, 13-7: 346-350 (September 1989).

66. Winters, E., "Requirements Checklist for A System Development Workstation," ACM SIGSOFT Software Engineering Notes, 11-5: 57-62 (October 1986).

67. Zucconi, L., "Selecting a CASE Tool," ACM SIGSOFT Software Engineering Notes, 14-2: 42-44 (April 1989).

Vita

Captain Jody L. Mattingly [redacted] and enlisted in the United States Marine Corps in November of that year. After serving four years as an aircraft avionics equipment technician, he enlisted in the United States Air Force (USAF) as an aircraft radar technician. In 1983 he was accepted under the Airman Education and Commissioning Program and attended Wright State University, from which he received the degree of Bachelor of Science in Computer Science in March, 1986. Upon graduation he received a regular commission through the USAF Officer Training School where he was a distinguished graduate. He then served as a system analyst as a member of the Headquarters Strategic Communications Division at Offutt Air Force Base, Omaha Nebraska. In May 1990, he received the degree of Master of Science in Mathematics from Creighton University. He entered the School of Engineering, Air Force Institute of Technology that same month.

[redacted]

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY (Leave blank)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	December]99]	Master's Thesis

4. TITLE AND SUBTITLE Establishing a Methodology for Evaluating and Selecting Computer Aided Software Engineering Tools for a Defined Software Engineering Environment at the Air Force Institute of Technology School of Engineering

5. FUNDING NUMBERS

6. AUTHOR(S)

Jody L. Mattingly, Captain, USAF

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

Air Force Institute of Technology, WPAFB OH, 45433-6583

8. PERFORMING ORGANIZATION REPORT NUMBER

AFIT GCS/ENG/9]D-]3

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION/AVAILABILITY STATEMENT

Approved for public release; distribution unlimited

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

This study identified the software engineering environment (SEE) as it exists at the Air Force Institute of Technology (AFIT) School of Engineering. It also desribes the software process model employed and the software development methods presented as part of the curriculum. Based on this information, criteria was established to evaluate computer aided software engineering (CASE) tools being considered for integration into the SEE. Each criterion was weighted to indicate its importance when selecting CASE tools. The criteria were further used to establish a methodology to be used to evaluate and select the CASE tools under consideration as well as future tool candidates.

14. SUBJECT TERMS			15. NUMBER OF PAGES
CASE, Software Engineering			]84
			16. PRICE CODE
17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified	Unclassified	UL